

BÁO CÁO ĐỒ ÁN

MÔN MẠNG MÁY TÍNH

1. Thông tin nhóm

MSSV	Họ và tên
19120176	Lê Công Bình
19120376	Nguyễn Lê Bảo Thi
19120036	Nguyễn Đăng Tiến Thành

2. Kịch bản giao tiếp

Client và Server kết nối với nhau thông qua giao thức TCP/IP. Sau khi Client gửi yêu cầu và kết nối thành công với Server, Server sẽ bắt đầu lắng nghe yêu cầu từ Client và thực hiện các chức năng tương ứng dựa trên thông điệp.

- Client gửi thông điệp đến Server dưới dạng chuỗi có nghĩa, với định dạng:

Chức năng, tác vụ, tham số 1, tham số 2, ...

Trong đó: - Các *tham số* trong lệnh cách nhau bởi dấu phẩy

- **Chức năng** là chức năng mà Client yêu cầu Server thực hiện. Như: *screenshot* (livestream và chụp ảnh), *shutdown* (tắt máy), *logoff* (logout máy) *keystroke* (keylogger), *process* (quản lý tiến trình), *application* (quản lý ứng dụng), *mac* (xem địa chỉ MAC), *folder* (xem cây thư mục)

- **Tác vụ:** là nội dung cụ thể của chức năng. Như: trong chức năng *process* sẽ có các tác vụ *view* (xem danh sách các tiến trình đang chạy), *kill* (diệt một tiến trình), *start* (bắt đầu một tiến trình).
 - *Tham số 1, tham số 2, ...* là danh sách các tham số cần thiết tùy thuộc vào chức năng tương ứng
- Server nhận thông điệp từ Client, giải mã và thực hiện gọi các API đã được cài đặt trong chương trình để thực hiện yêu cầu tương ứng từ phía Client. Sau đó đóng gói dữ liệu kết quả và gửi về Client, hoặc báo lỗi nếu phát sinh.
 - Client nhận dữ liệu kết quả từ Server gửi về và thực hiện giải mã, hiển thị lên giao diện người dùng.

3. Môi trường, thư viện hỗ trợ, cài đặt

A. Môi trường

Phần mềm hỗ trợ chạy trên môi trường hệ điều hành Windows 7 trở lên.

B. Thư viện hỗ trợ

Thư viện	Công dụng	Tài liệu
tkinter	Tạo giao diện người dùng	tkinter — Python interface to Tcl/Tk — Python 3.10.0 documentation
socket	Giao tiếp giữa client và server	socket — Low-level networking interface — Python 3.10.0 documentation
os	Chạy các lệnh điều khiển hệ thống như shutdown, logoff, xem tiến trình và ứng dụng	os — Miscellaneous operating system interfaces — Python 3.10.0 documentation

pynput	Lắng nghe sự kiện bàn phím	pynput · PyPI
keyboard	Khóa bàn phím	keyboard · PyPI
PIL	Chụp ảnh và live màn hình	Pillow · PyPI
winreg	Hỗ trợ các thao tác trên register	winreg — Windows registry access — Python 3.10.0 documentation
threading	Chia luồng	threading — Thread-based parallelism — Python 3.10.0 documentation
shutil	Copy, delete, move tập tin	shutil — High-level file operations — Python 3.10.0 documentation

C. Cài đặt

Để chạy mã nguồn ứng dụng, máy cần cài đặt Python 3 và một số thư viện cần thiết bằng cách chạy dòng lệnh

```
pip install -r requirements.txt
```

4. Hướng dẫn sử dụng

Chạy trực tiếp thông qua file thực thi ***server.exe*** và ***client.exe*** trong thư mục *Release*.

Người dùng cũng có thể chạy ứng dụng thông qua mã nguồn bằng dòng lệnh.

- Đối với Server, vào thư mục *code/server* và chạy lệnh

```
python main.py
```

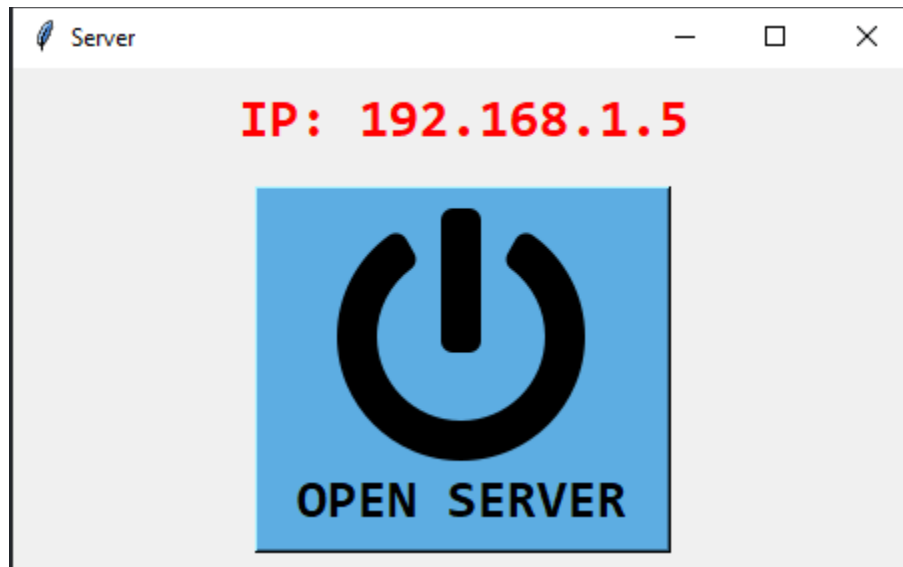
- Đối với Client, vào thư mục *code/client* và chạy lệnh

```
python main.py
```

5. Cách hoạt động

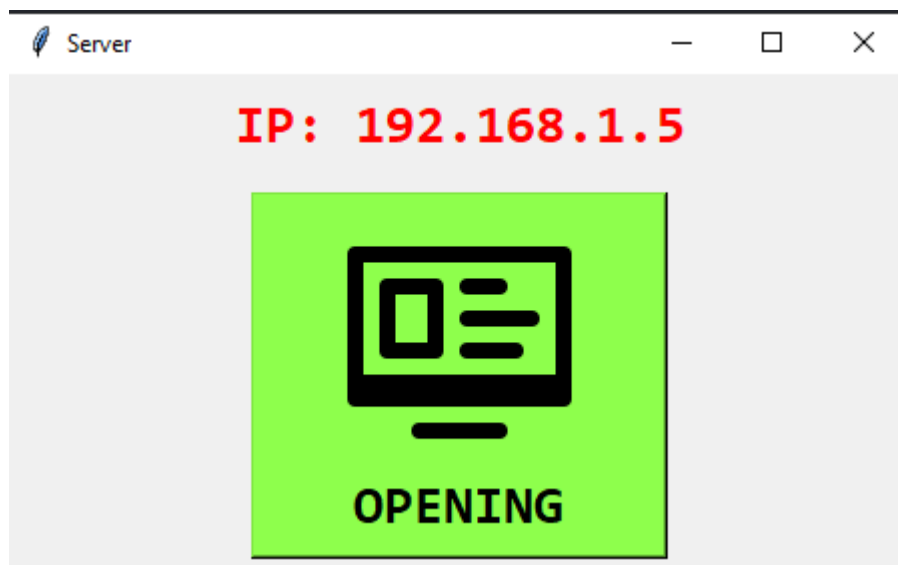
A. SERVER

Sau khi mở ứng dụng Server sẽ có giao diện như sau



Giao diện ban đầu hiển thị địa chỉ IPv4 và nút OPEN SERVER

Khi người dùng nhấn vào OPEN SERVER, Server sẽ gọi hàm `open_close_server(self)` để bắt đầu lắng nghe kết nối từ client. Dưới đây là giao diện sau khi mở Server:



Sau khi nhận được yêu cầu kết nối từ Client, Server sẽ gọi hàm `handle_client(self)` bắt đầu vào quá trình nhận thông điệp và xử lý yêu cầu từ Client.

Các chức năng ở Server được cài đặt riêng thành từng file theo phương pháp hướng đối tượng trong thư mục `code/server/`. Dưới đây là những chức năng được cài đặt ở Server để đáp ứng yêu cầu từ Client.

a. Keylogger – bắt phím nhấn, khóa bàn phím

Được cài đặt trong file `keystroke.py` và gồm các chức năng chính:

- Bắt đầu lắng nghe sự kiện bàn phím

```
def hook_key(self)
```

- Kết thúc lắng nghe bàn phím

```
def unhook_key()
```

- Lấy dữ liệu đã lắng nghe được và gửi về client

```
def print_keys(self)
```

- Khóa bàn phím

```
def lock_keyboard(self)
```

- Mở khóa bàn phím

```
def unlock_keyboard(self)
```

b. Livestream màn hình

Được cài đặt trong file `screenshot.py` gồm 2 chức năng chính:

- Xem live màn hình, thực hiện liên tục chụp ảnh màn hình và đóng gói gửi về Client, cho đến khi Client yêu cầu dừng
- Chụp ảnh màn hình tại bất kì thời điểm nào

```
def capture(self)
```

c. Shutdown và Logout

Được cài đặt trong file *server.py*

- Tắt máy tính sau 10 giây bằng cách gọi lệnh “shutdown -s” thông qua command line

```
def shutdown(self)
```

- Logout bằng cách gọi lệnh “shutdown -l” thông qua command line

```
def logoff(self)
```

d. Xem MAC address

Được cài đặt trong file *server.py*, thực hiện gọi các API để lấy địa chỉ MAC, ngoài ra còn có các thông tin khác của hệ thống như tên hệ điều hành, phiên bản hệ điều hành, tên thiết bị, processor, kiến trúc thiết bị.

```
def get_MAC_address(self)
```

e. Quản lý tập tin, cây thư mục

Được cài đặt trong file *folder.py*, gồm có các chức năng:

- Xem cây thư mục, cấu trúc của bất kì thư mục nào, đóng gói và gửi danh sách folder và file trong thư mục được Client yêu cầu

```
def view_folder(self, path)
```

Trong đó: path là đường dẫn mà Client yêu cầu được xem cấu trúc

- Xóa file hoặc folder bất kì

```
def delete_file(self, path)
```

Trong đó: path là đường dẫn của folder hoặc file mà Client yêu cầu xóa

- Copy file bất kì trong Server, gửi một file bất kì đến Client và nhận một file bất kì từ Client

```
def copy_file(self, source, target)
```

Trong đó: *source* là đường dẫn của file nguồn, *target* là đường dẫn của file đích. Nếu *source* = "?" thì Server sẽ nhận file từ Client và lưu tại *target*, nếu *target* = "?" thì Server sẽ gửi file ở *source* đến Client

- Di chuyển file bất kì trong Server

```
def move_file(self, source, target)
```

Trong đó: *source* là đường dẫn file nguồn cần di chuyển và *target* là đường dẫn đến folder cần di chuyển đến

f. Quản lý tiến trình

Cài đặt trong file *process.py*, gồm có 3 hàm chính

- Lấy toàn bộ tiến trình hiện có và gửi sang Client

```
def process_view(self)
```

- Diệt một tiến trình bất kỳ thông qua process ID

```
def process_kill(self, pid)
```

- Bắt đầu một tiến trình bất kỳ thông qua tên

```
def process_start(self, pname)
```

g. Quản lý ứng dụng

Cài đặt trong file *application.py*, gồm có 3 hàm chính:

- Lấy toàn bộ ứng dụng đang chạy và gửi sang Client

```
def application_view(self)
```

- Diệt một ứng dụng bất kỳ thông qua process ID

```
def application_kill(self, pid)
```

- Bắt đầu một ứng dụng bất kỳ thông qua tên

```
def application_start(self, pname)
```

h. Registry

Cài đặt trong file *registry.py*, gồm các chức năng chính liên quan đến quản lý registry

- Nhận dữ liệu file .reg từ Client gửi đến và đưa vào registry

```
def update_file_reg(self)
```

- Lấy giá trị của một value trong khóa cụ thể

```
def get_registry(self, reg, link, name)
```

- Tạo một khóa mới

```
def create_registry(self, reg, link)
```

- Gán giá trị cho một value trong khóa cụ thể

```
def set_registry(self, reg, link, name, datatype, value)
```

- Xóa một value trong khóa cụ thể

```
def delete_value_registry(self, reg, link, name)
```

- Xóa một khóa cụ thể

```
def delete_key_registry(self, reg, link)
```

B. CLIENT

Các chức năng được cài đặt riêng thành từng file theo phương pháp lập trình hướng đối tượng và được đặt trong thư mục code/client. Dưới đây là những hàm được Client gọi để gửi yêu cầu tới Server và xử lý kết quả trả về.

a. Socket kết nối đến Server

Cài đặt trong file *mysocket.py*, gồm có 1 class MySocket. Đối tượng MySocket cung cấp các hàm giúp thao tác nhận và gửi dữ liệu được dễ dàng hơn.

Các phương thức quan trọng gồm có:

- Kết nối tới máy chủ:

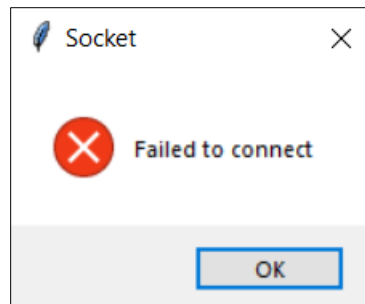
```
def connect(self, ip, port=54321)
```

Trong đó: *ip* là địa chỉ IPv4 của server, *port* được mặc định là 54321.

- Gửi yêu cầu tới máy chủ:

```
def send(self, msg="exit", showerror=True)
```

Trong đó: *msg* là yêu cầu được gửi tới server, *showerror* nếu là True thì khi có lỗi xảy ra lúc gửi yêu cầu (ví dụ như server bị tắt hoặc mất kết nối) thì sẽ hiện ra messagebox báo lỗi, còn bằng False thì sẽ không hiện thông báo.



- Gửi yêu cầu tới máy chủ không bao gồm header được sử dụng để gửi các yêu cầu tới server:

```
def send_immediate(self, msg="exit")
```

Trong đó: *msg* là yêu cầu được gửi tới server.

- Nhận dữ liệu với kích thước biết trước:

```
def receive(self, callback=None)
```

Trong đó: *callback* hàm callback được truyền vào để cập thành progress bar.

b. Utilities

Cài đặt trong file *utils.py*, gồm có 1 class *inputbox*, 1 class *ProgressBar*, và một hàm *messagebox*. Đây là các hàm hỗ trợ được gọi bởi các lớp khác để hiện thông báo hoặc lấy input từ người dùng,...

Các phương thức, class quan trọng gồm có:

- Gọi *messagebox*: Thông báo cho người dùng khi đã thao tác thành công hoặc bị lỗi:

```
def messagebox(title="client", msg="Done", type="info")
```

Trong đó: *title* là tiêu đề của cửa sổ, cho biết thông báo thuộc chức năng nào, *msg* là nội dung hiển thị trong thông báo, *type* là loại thông báo, gồm có “info”, “error” và “warning”.

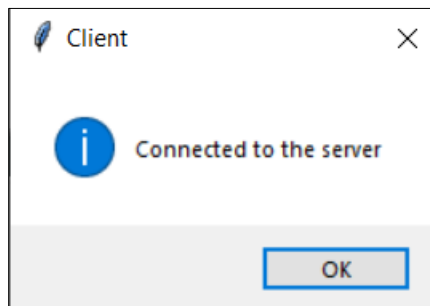


Figure 1: Info

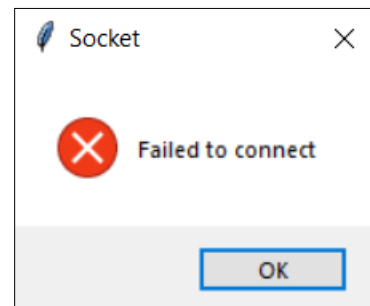


Figure 2: Error

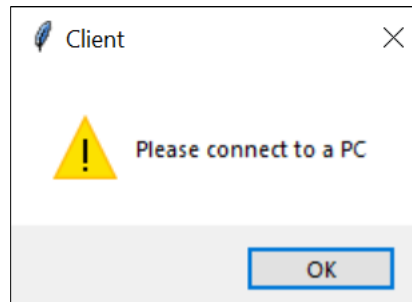


Figure 3: Warning

- Hiện cửa sổ cho phép nhập input, hỗ trợ chủ yếu cho class Manager của chức năng quản lý tiến trình và quản lý ứng dụng:

```
class inputbox(tk.Frame)
```

- Hiện thị thanh tiến độ khi truyền dữ liệu:

```
class ProgressBar(tk.Frame)
```

c. Header và Controller



Được cài đặt trong *rootview.py* gồm có giao diện header, và xử lý logic cho client bao gồm việc gửi, nhận thông tin và bind các phương thức của nó với các nút trên cửa sổ.

Phương thức quan trọng gồm:

- Khởi tạo header gồm các thành phần UI tên project, khung nhập địa chỉ IP và nút Connect/Disconnect để kết nối tới Server.

```
def create_header(self)
```

- Chạy chương trình, được gọi bởi hàm *main* để chạy toàn bộ client của ứng dụng:

```
def run(self)
```

- Kết nối tới server:


```
def connect(self)
```

- Ngắt kết nối tới server:

```
def disconnect(self)
```

- Thoát chương trình:

```
def exit_prog(self, e, isKilled=True)
```

Trong đó: *isKilled* giúp phân biệt khi ứng dụng được tắt bởi nút close window ở góc phải bên trên (*isKilled = True*) hay nút Exit ở dưới (*isKilled = False*). Lý do: nút  thực hiện sẵn việc hủy các thành phần UI, còn nút Exit được gọi hàm hủy UI một cách thủ công.



- Gọi cửa sổ các chức năng của chương trình: quản lý tiến trình, quản lý ứng dụng, keylogger, live server và chụp màn hình, quản lý tập tin, registry và thực hiện chức năng xem thông tin máy server, tắt máy server, log out, thoát chương trình:

```
def bind_actions(self)
```

- Thực hiện chức năng quay lại màn hình chính:

```
def back_to_menu(self)
```

- Thực hiện chức năng lấy thông tin của máy Server:

```
def get_info(self)
```

- Thực hiện chức năng tắt máy server:

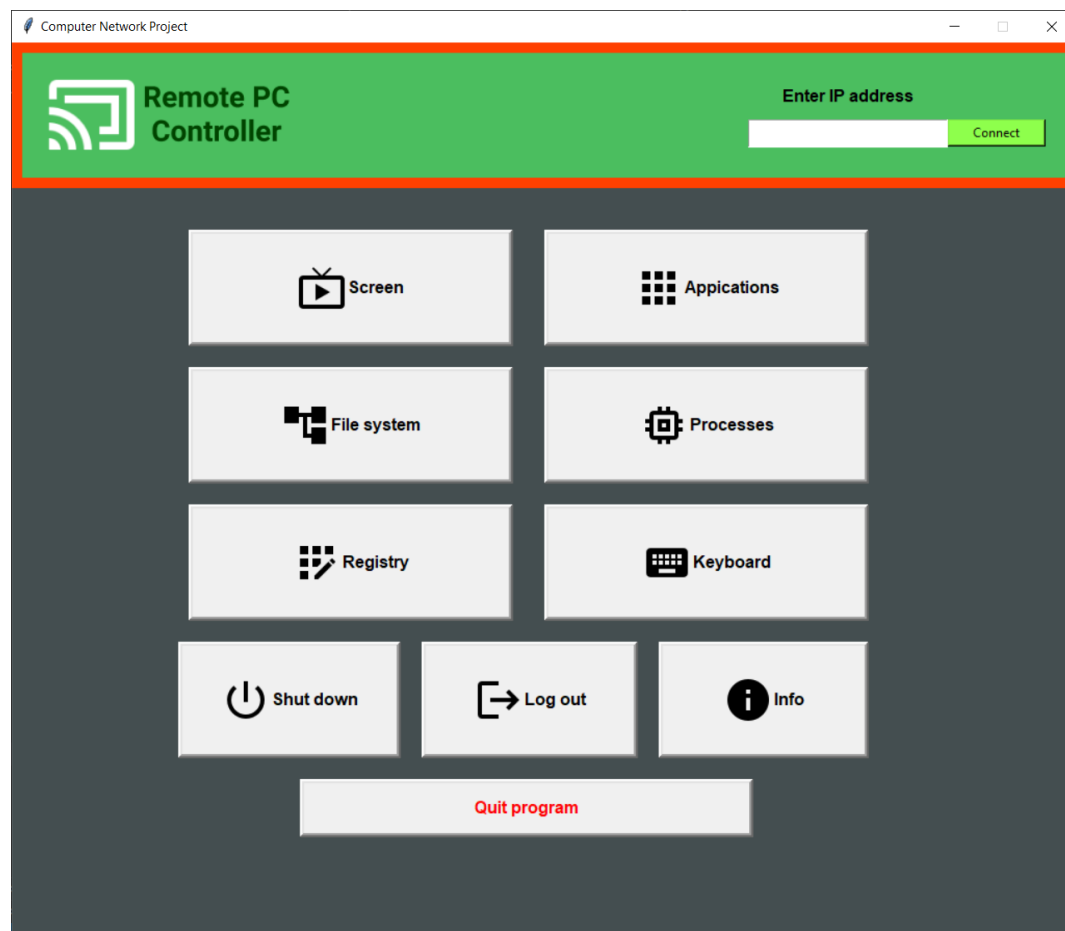
```
def shutdown(self)
```

- Thực hiện chức năng log out:

```
def logout(self)
```

d. Menu chính

Khi mở CLIENT thì giao diện menu chính của chương trình sẽ được hiển thị như sau:



Menu chính được cài đặt trong file *menu.py* gồm có 1 class Menu, dùng để tạo giao diện cho menu chính.

Phương thức quan trọng gồm có:

- Khởi tạo các thành phần nút của menu chính:

```
def create_widgets(self)
```

e. **Keylogger – bắt phím nhấn, khóa bàn phím**

Cài đặt trong file *keystroke.py* gồm có 1 class Keystroke, dùng để tạo giao diện giúp người dùng bắt tín hiệu bàn phím và khóa bàn phím của máy server.

Phương thức quan trọng gồm có:

- Tạo giao diện thực hiện keylogger:

```
def create_widgets(self)
```

- Thực hiện bắt bàn phím:

```
def keystroke_hook(self)
```

- In kết quả từ server lên màn hình:

```
def keystroke_print(self)
```

- Thực hiện khóa bàn phím:

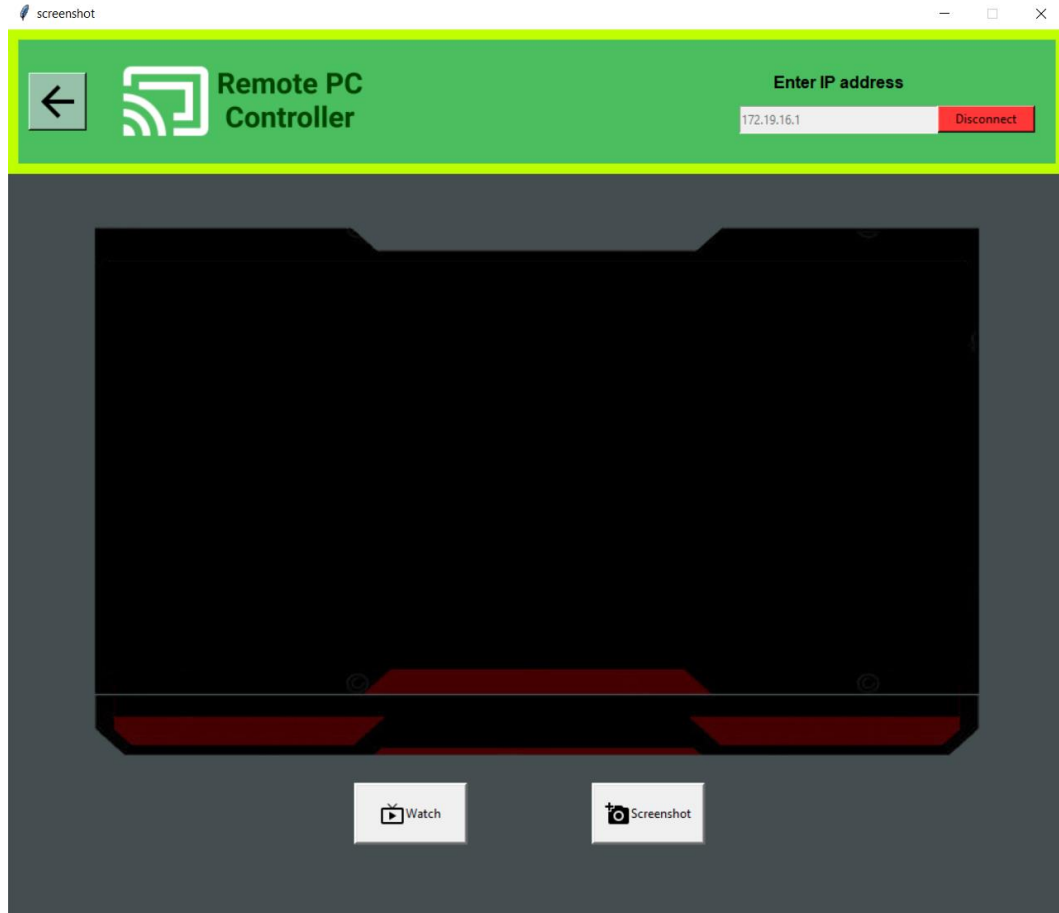
```
def keystroke_lock(self)
```

- Thực hiện xóa những hiển thị đã bắt được:

```
def keystroke_clear(self)
```

f. **Livestream màn hình**

Cài đặt trong file *screenshot.py*, gồm có 1 class Screenshot, dùng để tạo nên giao diện của cửa sổ xem live màn hình server kết hợp chức năng chụp ảnh màn hình server.



Các phương thức quan trọng gồm có:

- Tạo giao diện thực hiện livestream:

```
def create_widgets(self)
```

- Thay đổi kích thước ảnh cho vừa khung hình:

```
def _resize_image(self, IMG)
```

Trong đó: *IMG* là 1 đối tượng dữ liệu ảnh *PIL.Image* cung cấp bởi thư viện *PIL*.

- Cập nhật hình ảnh mới:

```
def update_image(self, img_data)
```

Trong đó: *img_data* là dãy bytes dữ liệu hình ảnh nhận trực tiếp từ server. Phương thức này trước khi render hình ảnh lên cửa sổ sẽ gọi gọi phương thức *_resize_image* để thay đổi kích thước

ảnh sao cho vừa với cửa sổ mà vẫn giữ đúng tỉ lệ. Đồng thời, phương thức cũng lưu dữ liệu ảnh vào thuộc tính *self._image_bytes*.

- Lưu ảnh vào máy:

```
def save_image(self)
```

Trong đó: Dữ liệu hình ảnh cần lưu có sẵn trong thuộc tính *self._image_bytes* nên khi gọi phương thức này không cần tuyên thêm biến dữ liệu hình ảnh.

- Thực hiện livestream màn hình và hình ảnh live được hiển thị trên màn hình:

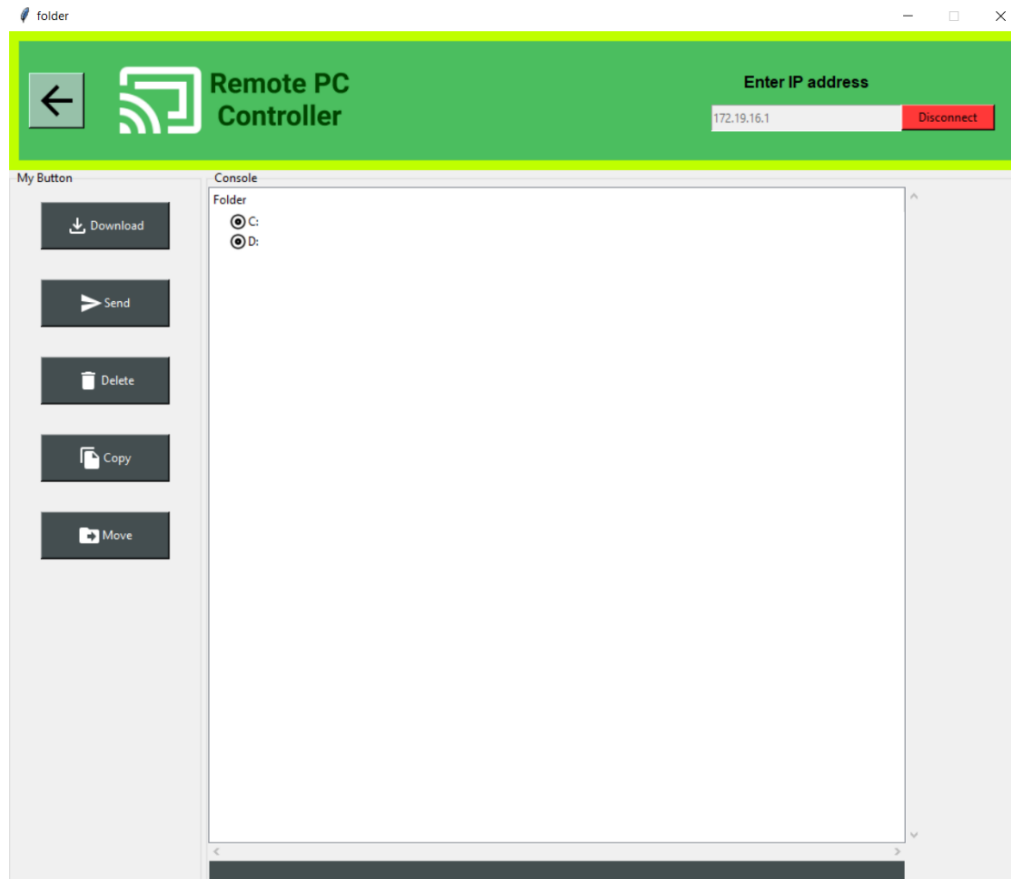
```
def stream_screen(self)
```

- Thực hiện chụp và mở hộp thoại lưu hình ảnh:

```
def snap_image(self)
```

g. Quản lý tập tin

Cài đặt trong file *filesystem.py*, gồm có 1 class *Filesystem*, dùng để tạo giao diện của cửa sổ của chức năng File System để hiển thị và quản lý các tập tin của server.



Các phương thức quan trọng gồm có:

- Khởi tạo các thành phần UI để thực hiện các chức năng và hiển thị File System:

```
def create_widgets(self)
```

- Thực hiện tải lại cây thư mục từ server:

```
def retrieve_file(self)
```

- Thực hiện gửi file từ client:

```
def send_file(self)
```

- Thực hiện xóa file:

```
def delete_file(self)
```

- Thực hiện copy file:

```
def copy_file(self)
```

- Thực hiện di chuyển file

```
def move_file(self)
```

- Thực hiện gửi:

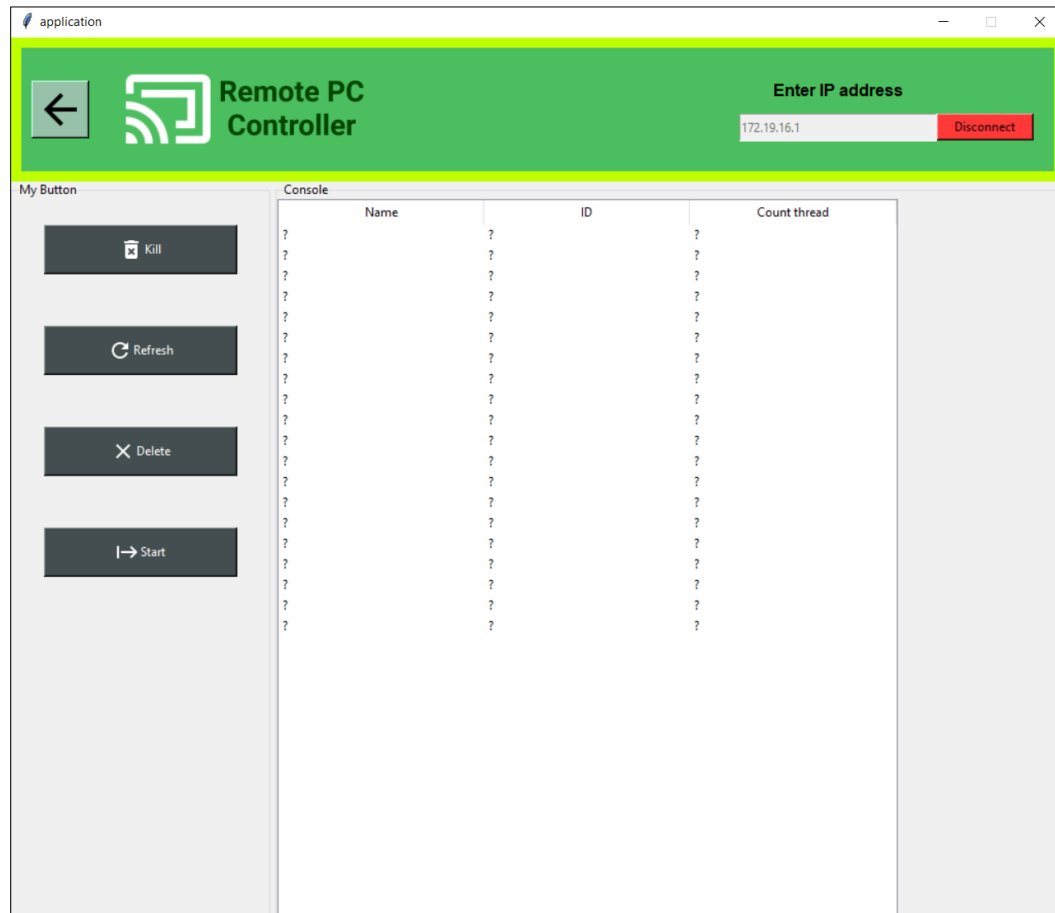
```
def send(self)
```

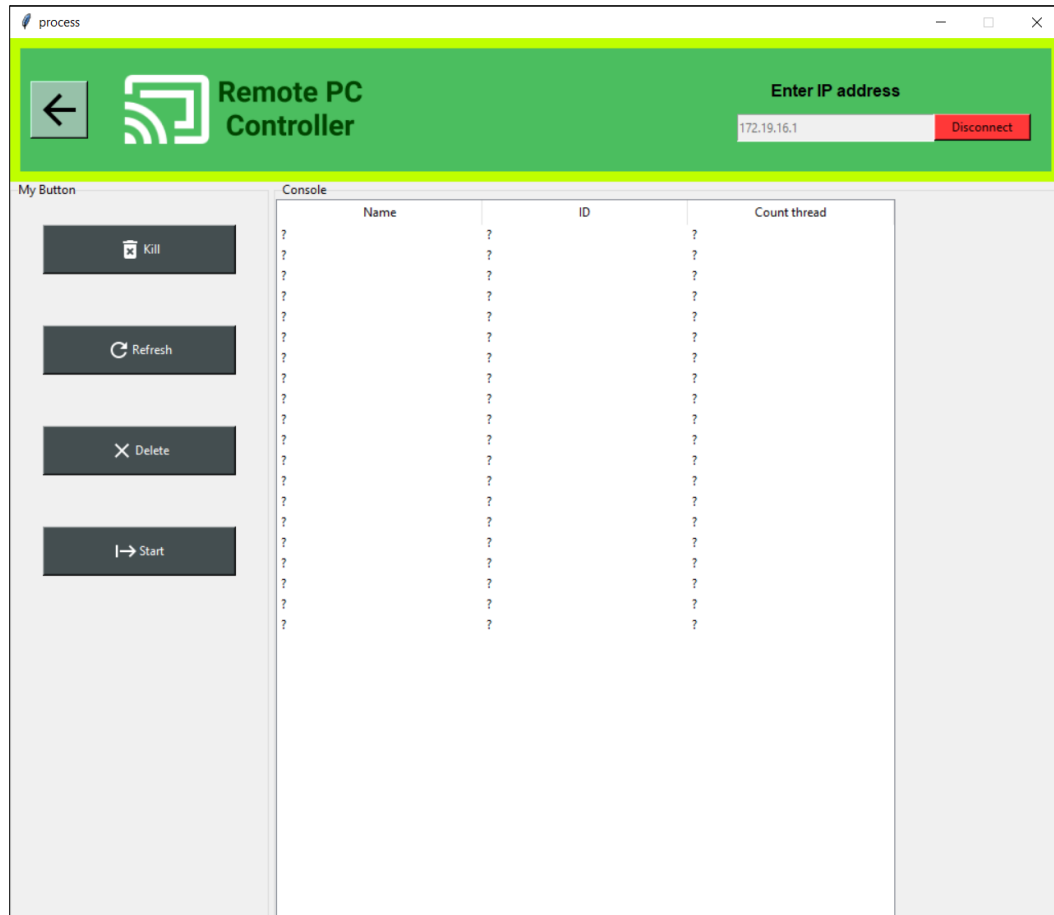
- Thực hiện nhận:

```
def receive(self)
```

h. Quản lý tiến trình, ứng dụng

Cài đặt trong file *manager.py*, gồm có 1 class Manager, dùng để tạo giao diện của cửa sổ của chức năng Process Running và App Running, bởi vì hai cửa sổ này có giao diện giống nhau.





Các phương thức quan trọng gồm có:

- Khởi tạo class Manager, tùy theo tham số truyền vào mà cửa sổ hiện ra sẽ là cửa Running process hoặc Running applications:

```
def __init__(self, parent, type)
```

Trong đó: *master* là một UI container trong thư viện *tkinter* của *python*, *command* cho biết cửa sổ được mở là cửa *Running process* hay là *Running application*. Nếu *command* là “*process*” thì sẽ là chức năng xem các tiến trình, nếu là “*application*” thì sẽ là tính năng xem các ứng dụng.

- Khởi tạo các thành phần UI như button, label, text field:

```
def create_widgets(self)
```

- Thực hiện chức năng kill process/application:

```
def kill(self)
```

- Thực hiện chức năng start process/application:

```
def start(self)
```

- Thực hiện chức năng refresh và hiển thị list process/application:

```
def view_async(self)
```

- Thực hiện chức năng xóa list process/application:

```
def clear(self)
```

i. Tương tác với Registry của Server

Cài đặt trong file *registry.py*, gồm có 1 class Registry, dùng để tạo giao diện giúp người dùng đọc, tạo, xóa, sửa registry trên máy server.



Các phương thức quan trọng gồm có:

- Tạo giao diện thực hiện registry:
- Mở file registry theo đường link, bằng cách gọi hộp thoại open file của hệ điều hành:

```
def create_widgets(self)
```

```
def browse_path(self)
```

- Đưa nội dung của file registry được mở vào vùng text, có thể chỉnh sửa nội dung trước khi gửi:

```
def update_cont(self)
```

- Thay đổi giao diện tùy theo chức năng đã chọn ở phần Sửa giá trị trực tiếp:

```
def update_ui(self, a, b, c)
```

Trong đó: a, b, c dùng để nhận các tham số mà hàm *self_df_func.trace* cung cấp, nhưng chương trình không cần đến.

- Gửi nội dung file registry đã chọn lên server:

```
def registry_sendcont(self)
```

- Gửi các cài đặt registry lên server:

```
def registry_send(self)
```

6. Tài liệu tham khảo

- Python Network Programming – David M.Beazley
- [Stack Overflow](#)
- Giáo trình Môn Mạng Máy Tính - Trường Đại học Khoa học Tự nhiên