

# BÁO CÁO ĐỒ ÁN MÔN MẠNG MÁY TÍNH

## LẬP TRÌNH SOCKET

### I. Thông tin nhóm

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

### II. Kịch bản giao tiếp

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

- Client gửi thông điệp đến Server dưới dạng chuỗi có nghĩa đã được mã hóa sang byte, 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.

Ví dụ: *screenshot* (chụp ảnh màn hình), *shutdown* (tắt máy), *keystroke* (keylogger), *process* (quản lý tiến trình), *application* (quản lý ứng dụng), *registry* và *quit* (đóng kết nối).

- **tác vụ** là nội dung cụ thể của chức năng. Ví dụ: trong chức năng process, view để xem các tiến trình, kill để diệt tiến trình, start để bắt đầu một tiến trình, ....

- tham số 1, tham số 2, ... là danh sách các tham số (có thể không cần tùy vào chức năng và tác vụ).

- Server đọc thông điệp từ client và thực hiện gọi các API đã được cài đặt trong chương trình tương ứng với yêu cầu client gửi đến. Sau đó đóng gói dữ liệu thu được và gửi về Client, hoặc báo lỗi đến client nếu trong quá trình thực hiện chức năng xuất hiện lỗi.
- Client đọc và giải mã dữ liệu do server gửi đến, sau đó xuất dữ liệu trên ứng dụng (GUI) hoặc báo lỗi nếu có.

### III. Môi trường, thư viện, cách cài đặt

#### A. Môi trường

Phần mềm có khả năng chạy trên môi trường hệ điều hành Window 10. Để chạy mã nguồn python của project, máy cần cài đặt python 3 x64.

#### B. Thư viện

Các thư viện đã được sử dụng bao gồm:

Thư viện	Công dụng	Tài liệu
tkinter	Tạo giao diện, các dialog box cho phần mềm	<a href="https://docs.python.org/3.7/library/tkinter.html">https://docs.python.org/3.7/library/tkinter.html</a>
codecs	Đọc file .reg định dạng utf-16	<a href="https://docs.python.org/3/library/codecs.html">https://docs.python.org/3/library/codecs.html</a>

io	Đọc dữ liệu ảnh dưới dạng byte	<a href="https://docs.python.org/3/library/io.html">https://docs.python.org/3/library/io.html</a>
PIL	Client dùng để chuyển dữ liệu ảnh về dạng mà tkinter có thể biểu diễn được. Server dùng để chụp màn hình.	<a href="https://pillow.readthedocs.io/en/stable/installation.html">https://pillow.readthedocs.io/en/stable/installation.html</a>
os	Chạy các dòng lệnh để điều khiển máy trên server	<a href="https://docs.python.org/3/library/os.html">https://docs.python.org/3/library/os.html</a>
pynput	Lắng nghe sự kiện nhập từ bàn phím	<a href="https://pynput.readthedocs.io/en/latest/index.html">https://pynput.readthedocs.io/en/latest/index.html</a>
winreg	Xem, thêm, xóa, sửa các giá trị registry của máy chạy server	<a href="https://docs.python.org/3/library/winreg.html">https://docs.python.org/3/library/winreg.html</a>
threading	Chia luồng GUI và xử lý tín hiệu từ client.	<a href="https://docs.python.org/3/library/threading.html">https://docs.python.org/3/library/threading.html</a>

### C. Cách cài đặt

Trong các thư viện trên, chỉ có pynput và PIL là cần phải được cài đặt, các thư viện còn lại đều có sẵn trong python 3.

Để cài đặt các thư viện còn lại, có thể dùng pip để cài lần lượt pynput và PIL (hướng dẫn cài có trong phần link thư viện). Hoặc có thể vào trong thư mục Socket-Programming và chạy dòng lệnh sau:

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

## IV. Hướng dẫn sử dụng

Để sử dụng ứng dụng Server, chạy lệnh sau trên Command Line:

```
python code/server/main.py
```

Để sử dụng ứng dụng Client, chạy lệnh sau trên Command Line:

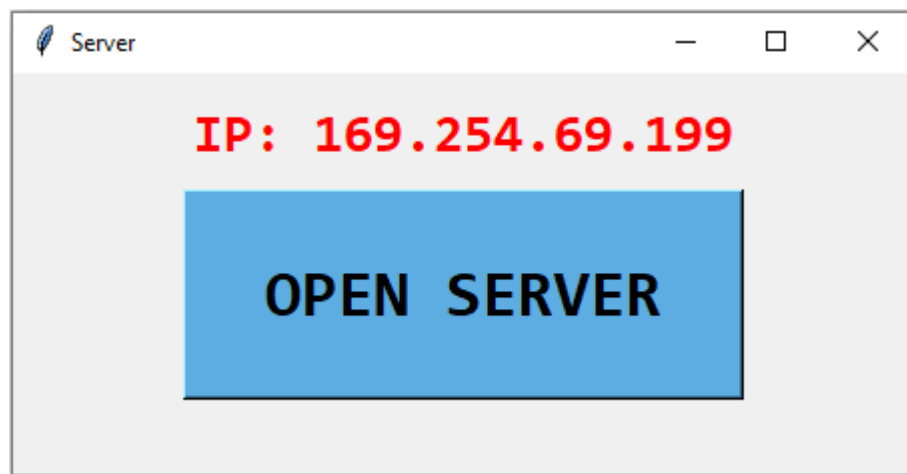
```
python code/client/main.py
```

Hoặc có thể chạy trực tiếp file thực thi: **server.exe** và **client.exe** trong thư mục *Release*.

## V. Cách hoạt động

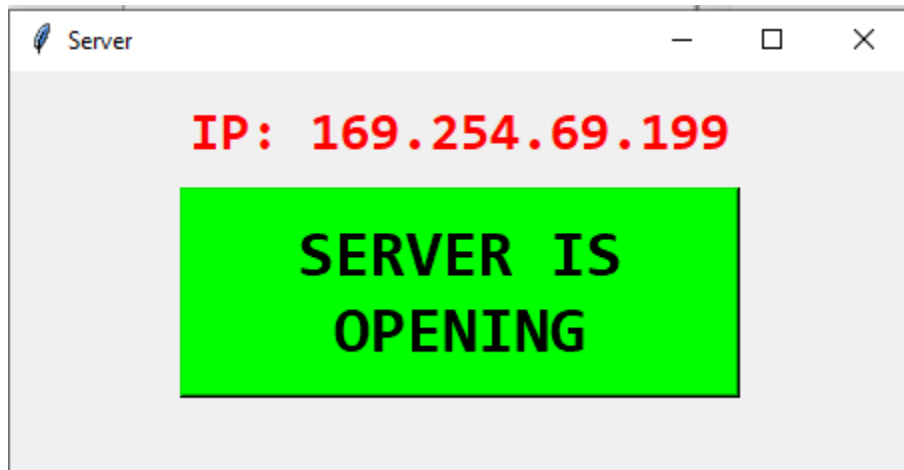
### A. SERVER

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



Giao diện hiển thị địa chỉ IPv4 của Server và nút OPEN SERVER.

Khi nhấn vào nút OPEN SERVER, server sẽ bắt đầu mở và lắng nghe kết nối từ *một client*. Dưới đây là giao diện sau khi mở server:



Sau khi nút OPEN SERVER được nhấn, hàm `open_close_server(self)` sẽ được gọi và tạo một socket server lắng nghe kết nối từ client.

Sau khi có client kết nối đến server, hàm `handle_client(self)` trong file `server.py` sẽ được gọi, để xử lý những thông điệp từ phía 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/server`. Dưới đây là những hàm được Server gọi để đáp ứng yêu cầu từ Client:

#### a) Quản lý tiến trình (Process)

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

- Xem toàn bộ tiến trình hiện có:

`process_view(self)`

- Diệt một tiến trình theo ID (pid):

`process_kill(self, pid)`

- Bắt đầu một tiến trình theo tên (pname):

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

## b) Quản lý ứng dụng (Application)

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

- Xem toàn bộ ứng dụng đang chạy:

```
application_view(self)
```

- Diệt một ứng dụng theo ID (pid):

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

- Bắt đầu một ứng dụng theo tên (pname):

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

## c) Chụp ảnh màn hình (Screenshot)

Cài đặt trong file *screenshot.py*, gồm hàm chính `run(self)` thực hiện chụp ảnh màn hình và nén dữ liệu về dạng byte gửi về client

## d) Keylogger

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

- Bắt đầu lắng nghe sự kiện bàn phím:

```
hook_key(self)
```

- Kết thúc lắng nghe:

```
unhook_key(self)
```

- Lấy dữ liệu đã lắng nghe và gửi về client dưới dạng chuỗi:

```
print_keys(self)
```

## e) Registry

Cài đặt trong file *registry.py*, hàm chính `run(self)` dùng để lắng nghe yêu cầu chỉnh sửa registry và xử lý tín hiệu từ client gửi đến. Gồm có 6 hàm chính:

- Nhận dữ liệu file .reg từ client và đưa vào registry:

```
update_file_reg(self)
```

- Lấy giá trị của một value trong khóa cụ thể:

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

**Trong đó:** *reg* là HKEY, *link* là đường dẫn đến khóa chứa value cần lấy giá trị, *name* chính là tên của value cần lấy.

- Tạo một khóa:

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

**Trong đó:** *reg* là HKEY, *link* là tên khóa cần tạo

- Gán giá trị cho một value trong khóa cụ thể:

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

**Trong đó:** *reg* là HKEY, *link* là đường dẫn đến khóa đến value cần gán giá trị, *name* là tên của value, *datatype* là kiểu dữ liệu của value được tạo và *value* là giá trị cụ thể cần gán cho value *name*.

- Xóa một value trong khóa cụ thể:

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

**Trong đó:** reg là HKEY, link là đường dẫn đến khóa, name là tên khóa cần xóa

- Xóa một khóa:

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

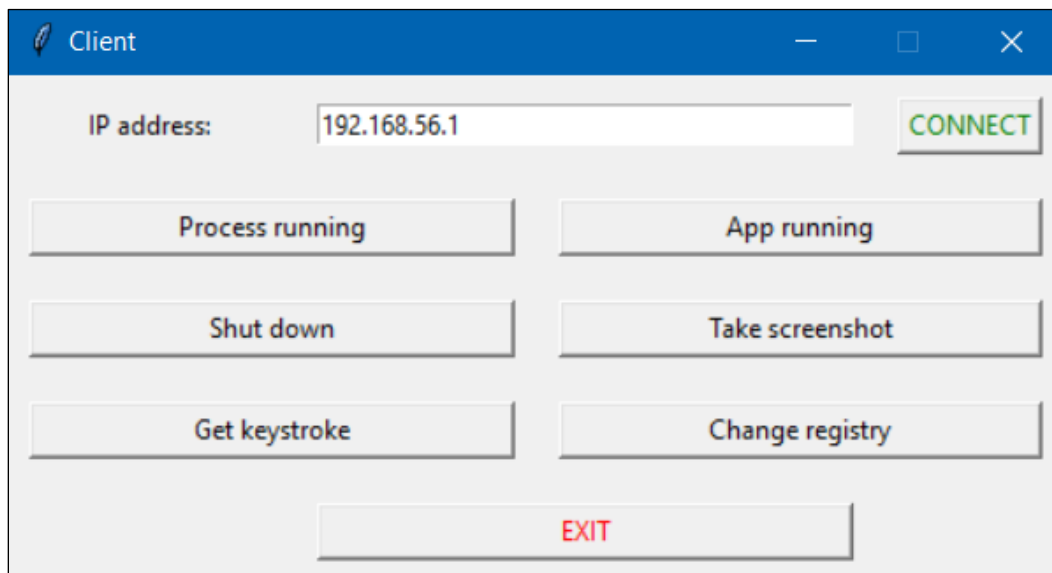
**Trong đó:** reg là HKEY, link là đường dẫn đến khóa cần xóa

## 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) Menu chính

Cài đặt trong file *menu.py*, gồm có 1 class Menu, dùng để tạo nên giao diện của menu chính.



Phương thức quan trọng gồm có:

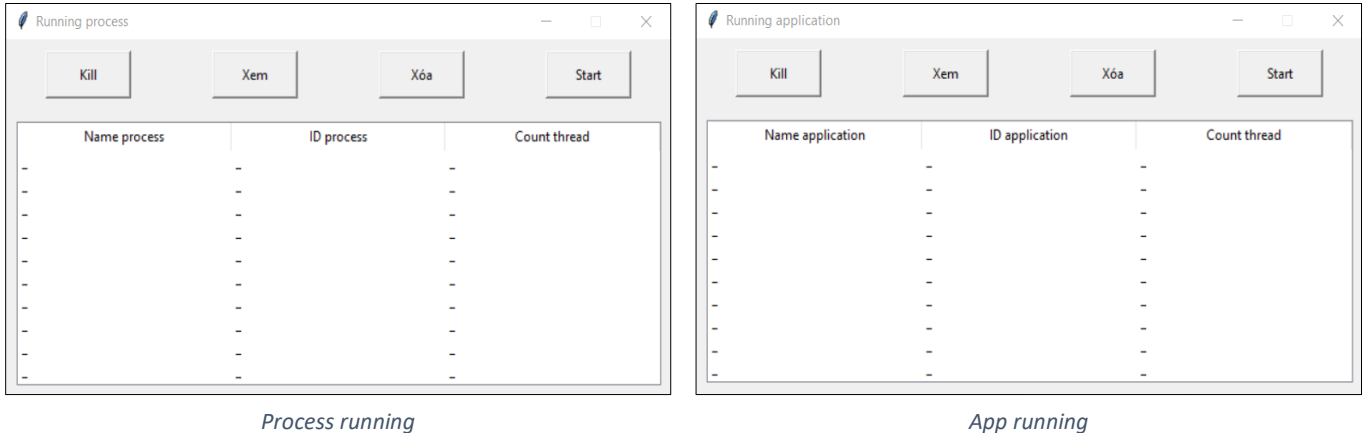
- Khởi tạo các thành phần UI như button, label, text field:

```
create_widgets(self)
```



## b) Quản lý tiến trình, ứng dụng

Cài đặt trong file *manager.py*, gồm có 1 class Manager, dùng để tạo nên 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 của *Running process* hoặc *Running application*:

```
__init__(self, master, command="process")
```

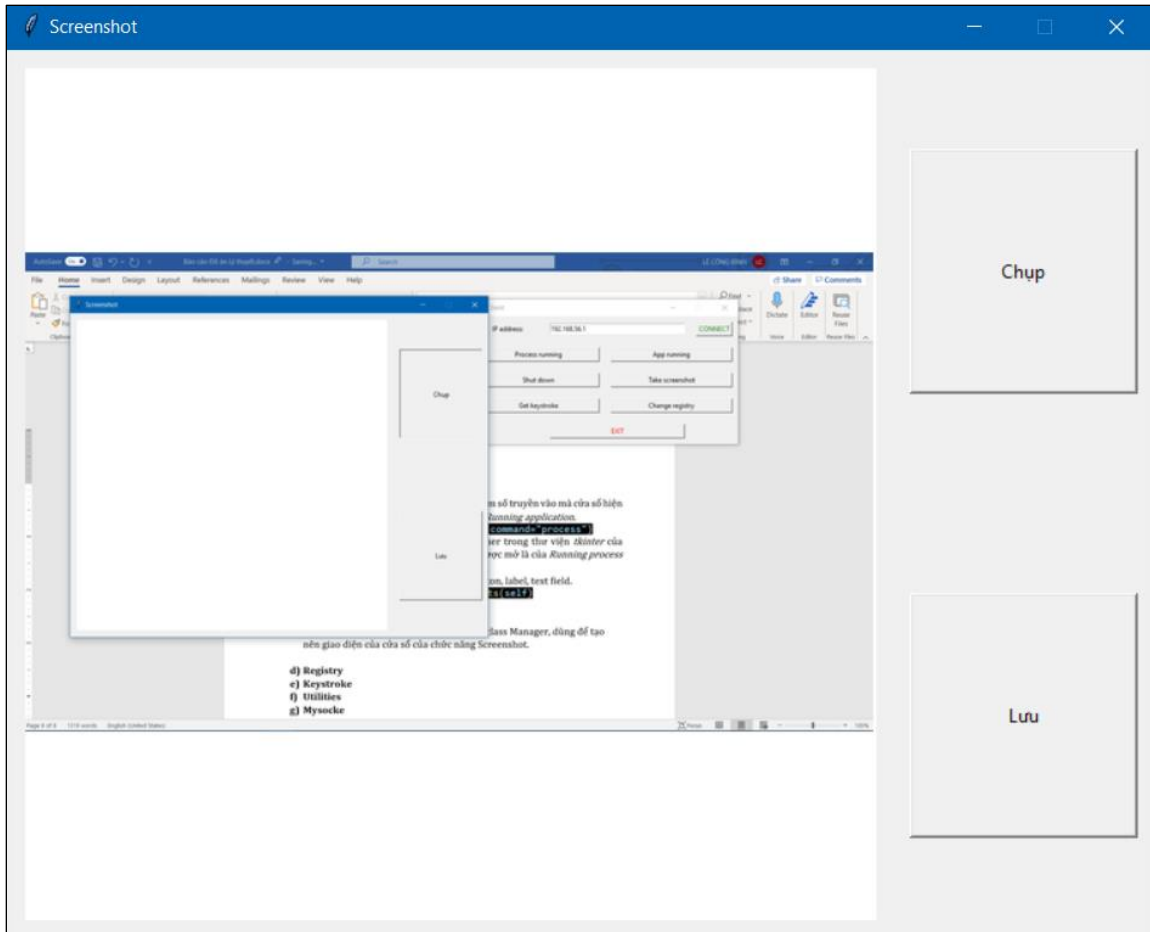
**Trong đó:** master là một UI container trong thư viện *tkinter* của *python*, command cho biết cửa sổ được mở là của *Running process* hay là *Running application*.

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

```
create_widgets(self)
```

## c) Chụp ảnh màn hình

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



Các phương thức quan trọng gồm có:

- Thay đổi kích thước ảnh cho vừa khung hình:

```
_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:

```
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 hàm `_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:

`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 truyền thêm biến dữ liệu hình ảnh.

#### d) Tương tác với Registry của server

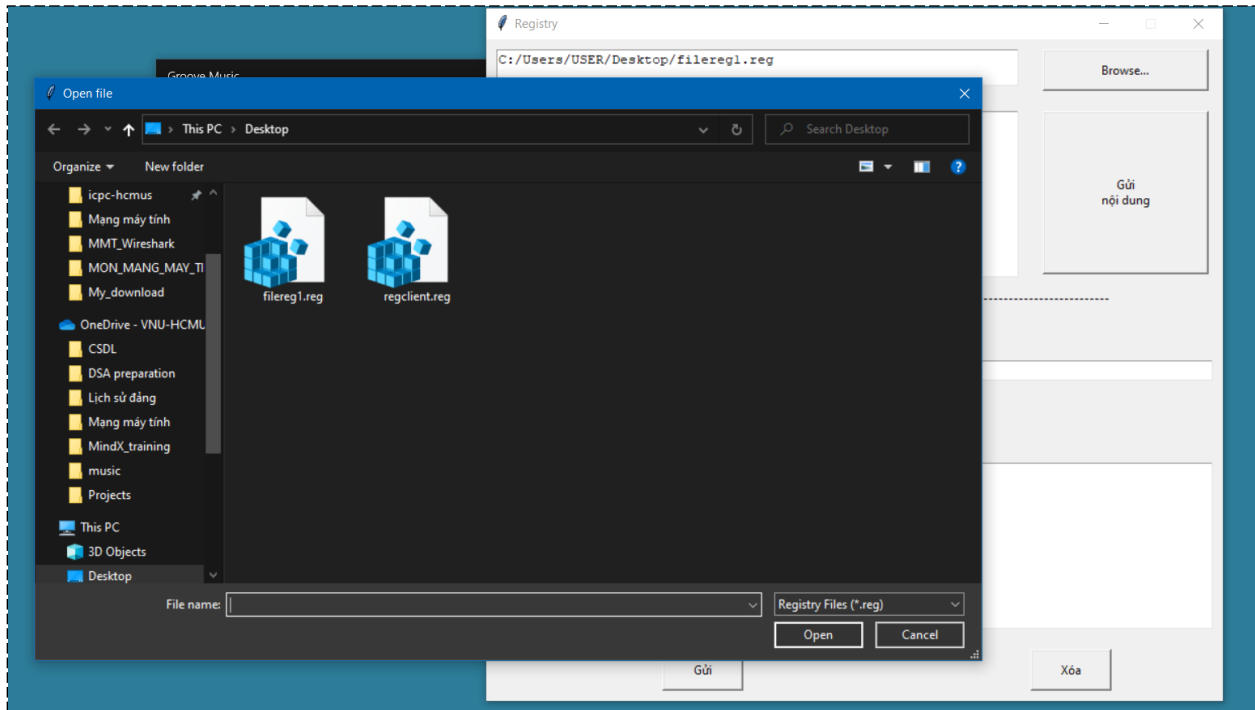
Cài đặt trong file *registry.py*, gồm có 1 class `Registry`, dùng để tạo giao diện giúp người dùng đọc, tạo, xóa, sửa registry trên máy server.

Các phương thức quan trọng gồm có:

- Mở file registry theo đường link, bằng cách gọi hộp thoại open file của hệ điều hành:

`browse_path(self)`

Ví dụ:



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

```
update_cont(self)
```

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

```
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.

Ví dụ:

Sửa giá trị trực tiếp

Chức năng:

Create key

Đường dẫn:

Sửa giá trị trực tiếp

Chức năng:

Set value

Đường dẫn:

Name value

Value

Data type

String

### e) Keylogger

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

Keystroke

Hook

Unhook

In phím

Xóa

<Key.shift>in ke6tl qua3 tu72 server le6n man2 hinh2.

Phương thức quan trọng gồm có:

- In kết quả từ server lên màn hình:

```
print_keystroke(self, keystroke="<Not hooked>")
```

**Trong đó:** keystroke là dữ liệu text nhận từ server khi nhấn nút *In phím*.

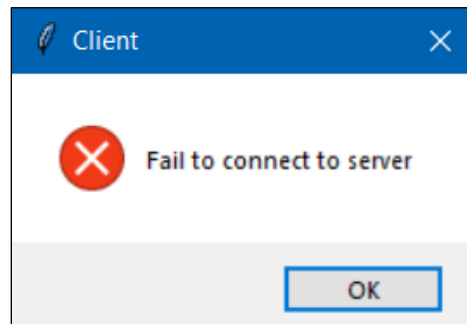
## f) Utilities

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

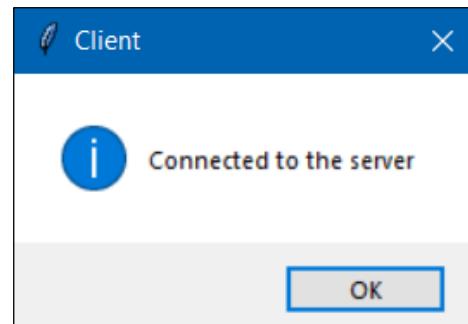
- Gọi *messagebox*, thông báo cho người dùng khi đã thao tác thành công hoặc bị lỗi:

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

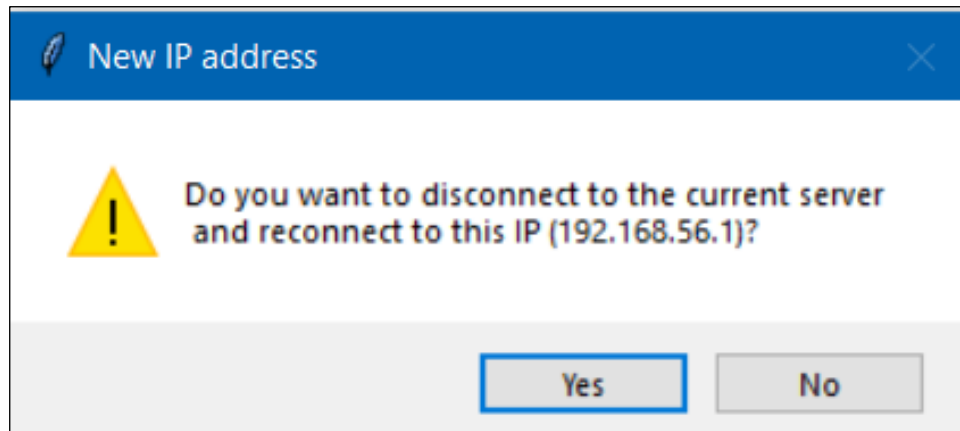
**Trong đó:** *title* là tiêu đề của cửa sổ, cho biết thông báo thuộc chức năng nào, *msg* là nội dung hiện thị trong thông báo, *type* là loại thông báo, gồm có “info”, “error” và “warning”.



*error*



*info*



*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)
```

### g) Socket kết nối tới 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:

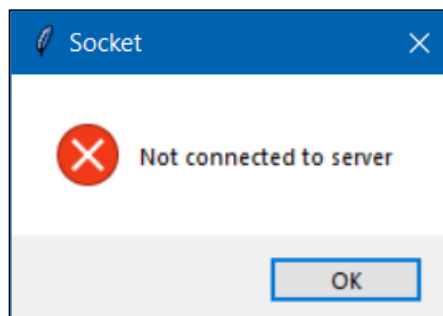
```
connect(self, ip, port=54321)
```

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

- Gửi yêu cầu tới máy chủ:

```
send(self, command="exit", showerror=True)
```

**Trong đó:** *command* 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.



- Nhận dữ liệu với kích thước biết trước:

```
receive(self, length = 2048)
```

**Trong đó:** *length* là kích thước dữ liệu được gửi đến. Với những thao tác mà dữ liệu gửi đến từ server có kích thước lớn, buộc phải nhận nhiều lần, như chụp ảnh màn hình, thì server sẽ gửi trước kích thước của dữ liệu, client sẽ nhận dữ liệu đó với *length* mặc định là 2048. Kích thước dữ liệu nhận được sẽ được dùng làm *length* lần nhận dữ liệu tiếp theo, khi mà sever gửi toàn bộ dữ liệu đã yêu cầu.

## h) Controller

Cài đặt trong file *controller.py*, gồm có 1 class *Controller*. Đối tượng *Controller* có trách nhiệm xử lý logic cho client, bao gồm việc gửi, nhận thông tin và bind các phương thức thức của nó với các nút trên cửa sổ.

Các phương thức quan trọng gồm có:

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

```
run(self)
```

- Kết nối tới server:


```
connect(self)
```

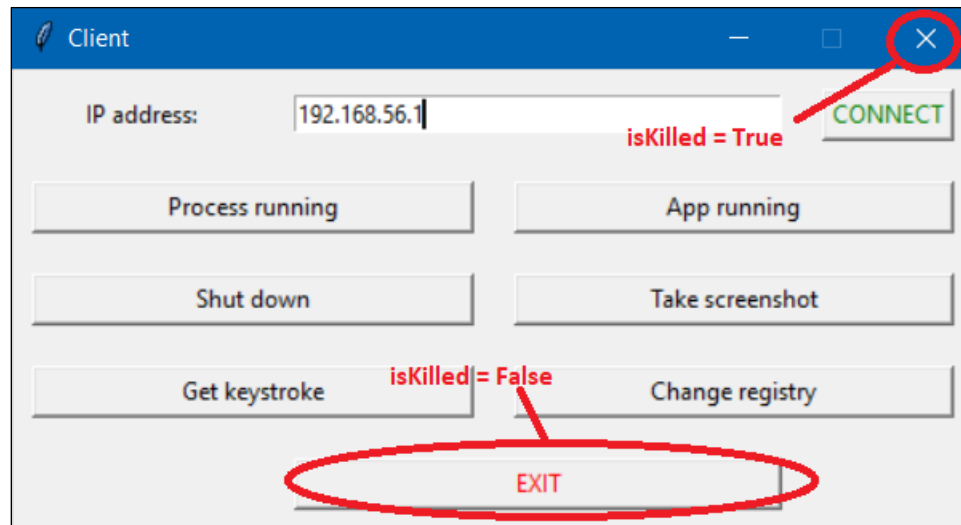
- Thoát chương trình:

```
exit_prog(self, isKilled=False)
```

**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). Lí do: nút  thực hiện sẵn việc hủy các thành phần UI, còn nút Exit phải gọi hàm hủy UI một cách thủ công.



- Gọi cửa sổ chức năng quản lý tiến trình:

`manager_prc(self)`

- Gọi cửa sổ chức năng quản lý ứng dụng:

`manager_app(self)`

- Gọi cửa sổ chức năng keylogger:

`keystroke(self)`

- Gọi cửa sổ chức năng chụp màn hình:

`screenshot(self)`

- Gọi cửa sổ chức năng registry:

`registry(self)`

- Thực hiện chức năng tắt máy server:

`shutdown(self)`

## VI. Tài liệu tham khảo

- [Stackoverflow.com](https://stackoverflow.com)
- Giáo trình mạng máy tính
- Python Network Programming – David M. Beazley

---

*HẾT*

---