Vietnam National University - Ho Chi Minh

University of Science
Faculty of Information Technology
.....o0o.....

Báo Cáo: Ứng Dụng Điều Khiển Từ Xa

Môn: Mạng máy tính

Theory teacher:	**Đỗ Hoàng Cường**	
Instructors practice:	**Nguyễn Thanh Quân**	
	Huỳnh Thụy Bảo Trân	
Student:	**Hoàng Hữu Minh An**	**20127102**
	Lê Phan Duy Tùng	**20127661**
	Thái Văn Thiên	**20127631**
Class:	**20CLC01**	

○ GROUP 07 ○

Mục lục:

I.Giời thiệu về đồ án:

Đồ án ứng dụng điều khiển từ xa được làm bởi nhóm 7 gồm các thành viên như sau:

1. Hoàng Hữu Minh An ID: 20127102
2. Lê Phan Duy Tùng ID: 20127661
3. Thái Văn Thiên ID: 20127631

Ứng dụng điều khiển từ xa của chúng em xin được báo cáo có những chức năng như sau:

a. Live Screen
b. Key logger
c. Directory tree
d. App Process
e. Registry
f. MAC address
g. ShutDown

II.Kịch bản giao tiếp của chương trình:

1) Kịch bản giao tiếp:

Chương trình ứng dụng điều khiển từ xa cho phép người dùng có thể điều khiển máy tính của mình trong cùng một mạng. Kết nối giữa server(máy chủ) và người dùng (client) thông qua địa chỉ IPV4 và cổng Port cố định, người dùng khi kết nối thành công phải đăng nhập bằng địa chỉ IPV4 của máy mà mình muốn điều khiển đã bật file server.exe. Sau khi kết nối thành công chương trình, người dùng có thể thao tác các công việc ra lệnh trên màn hình GUI của ứng dụng, server sẽ bắt được tin nhắn và sẽ thức hiện các thao tác và gửi đến cho client. Cụ thể, chương trình đã cung cấp các chức năng như sau: Live screen, Key logger, Directory tree, App process, Registry, MAC address, ShutDown

2) Giao thức trao đổi giữa client và server: TCP/IPv4

3) Chương trình gồm có các file:

- File: client.py
- File: server.py
- File: logInPage_GUI.py
- File: homePage_GUI.py
- File: live_screen_client.py, live_screen_server.py
- File: keylogger_client.py, keylogger_server.py

- File: directory_tree_client.py, directory_tree_server.py
- File: app_process_client.py, app_process_server.py
- File: registry_client.py, registry_server.py
- File: mac_address_client.py, mac_address_server.py
- File: shutdown_logout_client.py, shutdown_logout_server.py

4) Cấu trúc thông điệp:

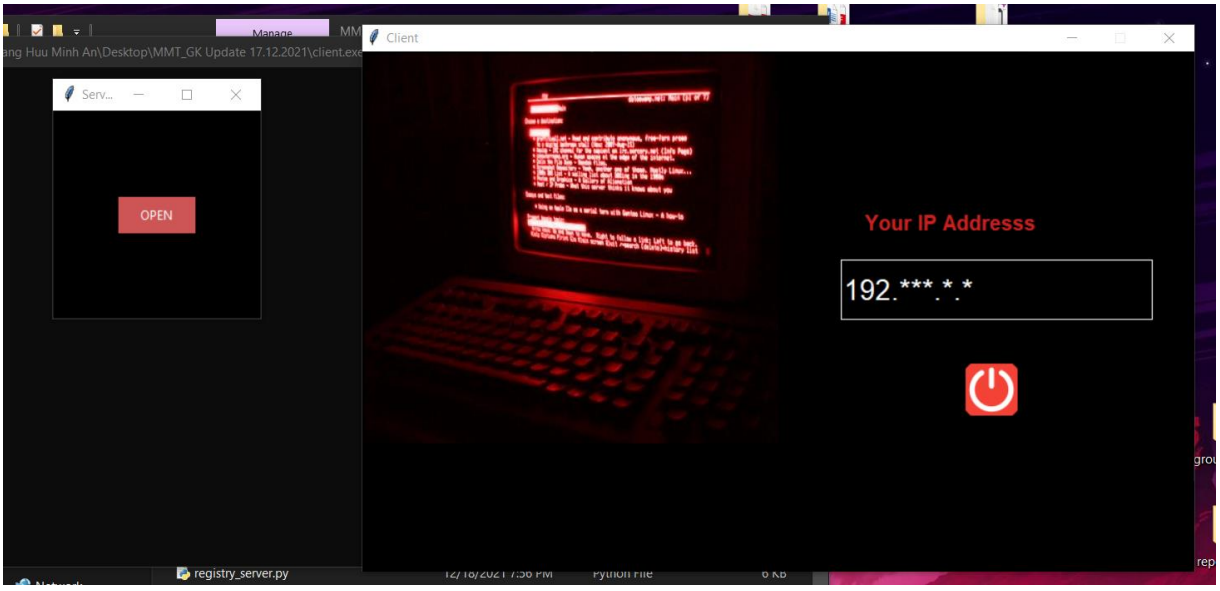
Để được phục vụ client sẽ gửi thông điệp yêu cầu tới server mô tả về công việc muốn server thực hiện. Khi nhận được request từ client thì server sẽ tiến hành phân tích và sử lý yêu cầu sau đó trả lời thông điệp cho client.

5) Kiểu dữ liệu của thông điệp:

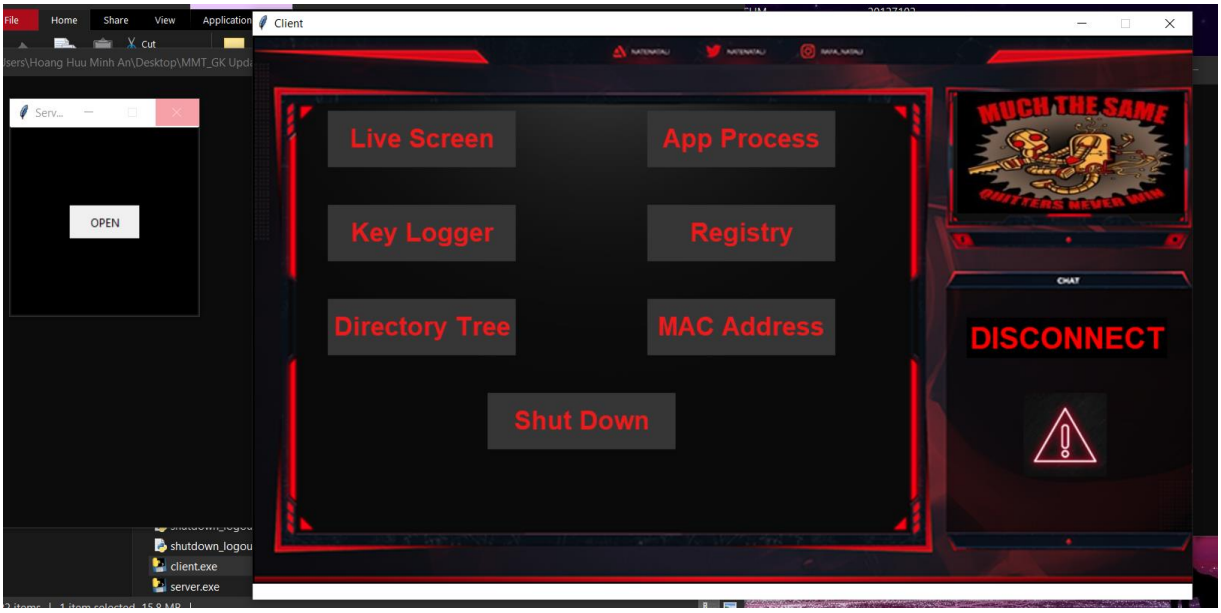
Thông điệp được mã hóa(encode) khi được gửi di và được giải mã(decode) khi nhận được để phân tích thông điệp.

III.Kịch bản chương trình:

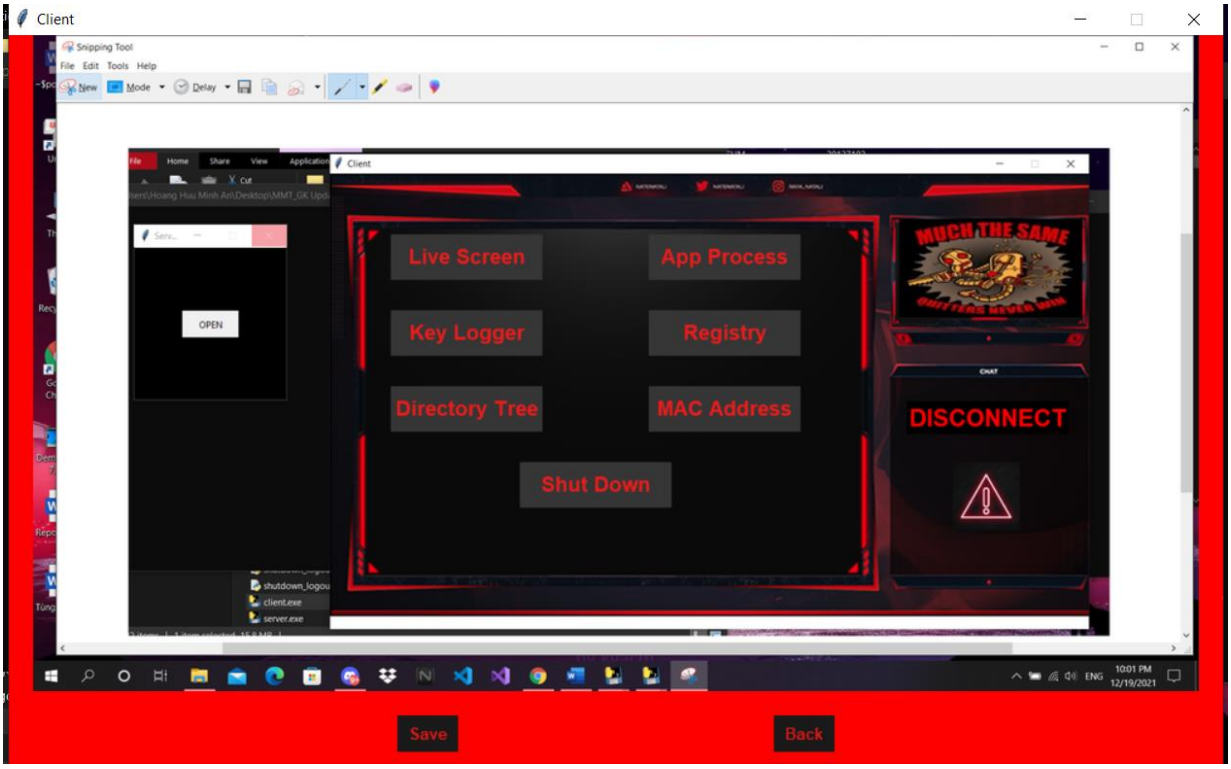
Giao diện khi mới khởi động chương trình và nhập địa chỉ IP



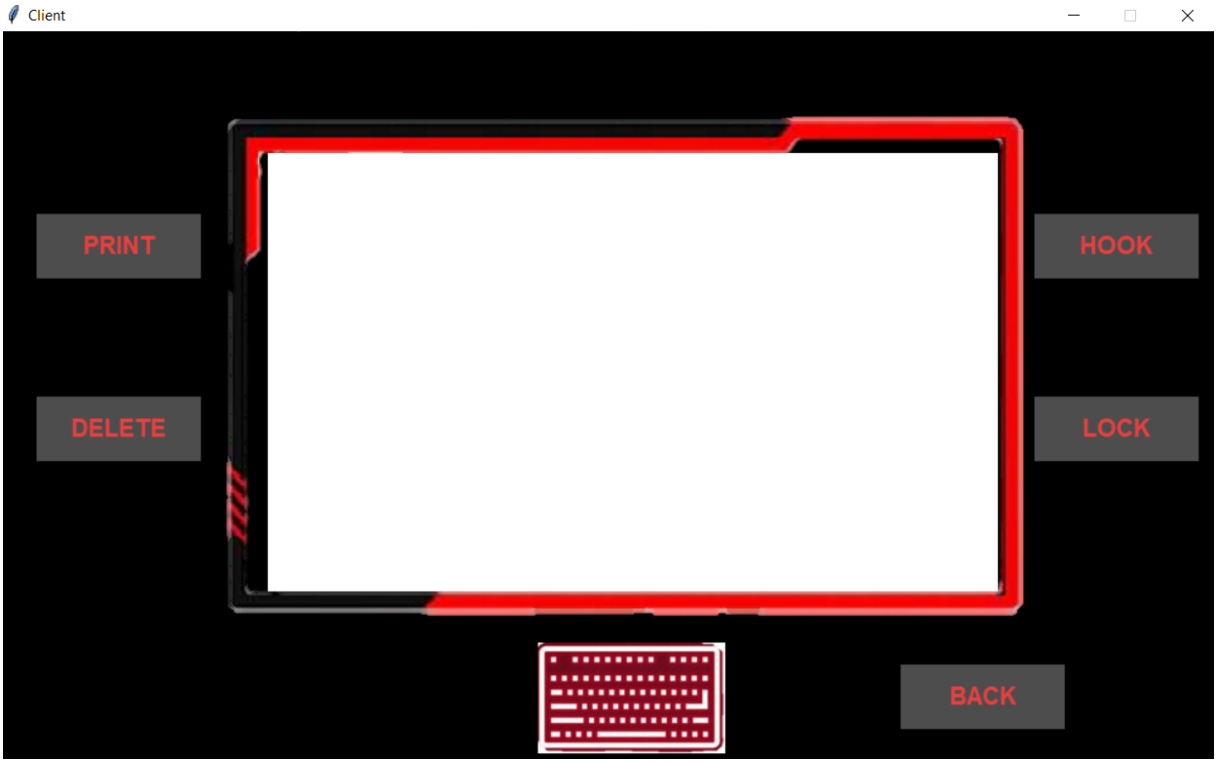
Giao diện màn hình điều khiển sau khi đã kết nối thành công



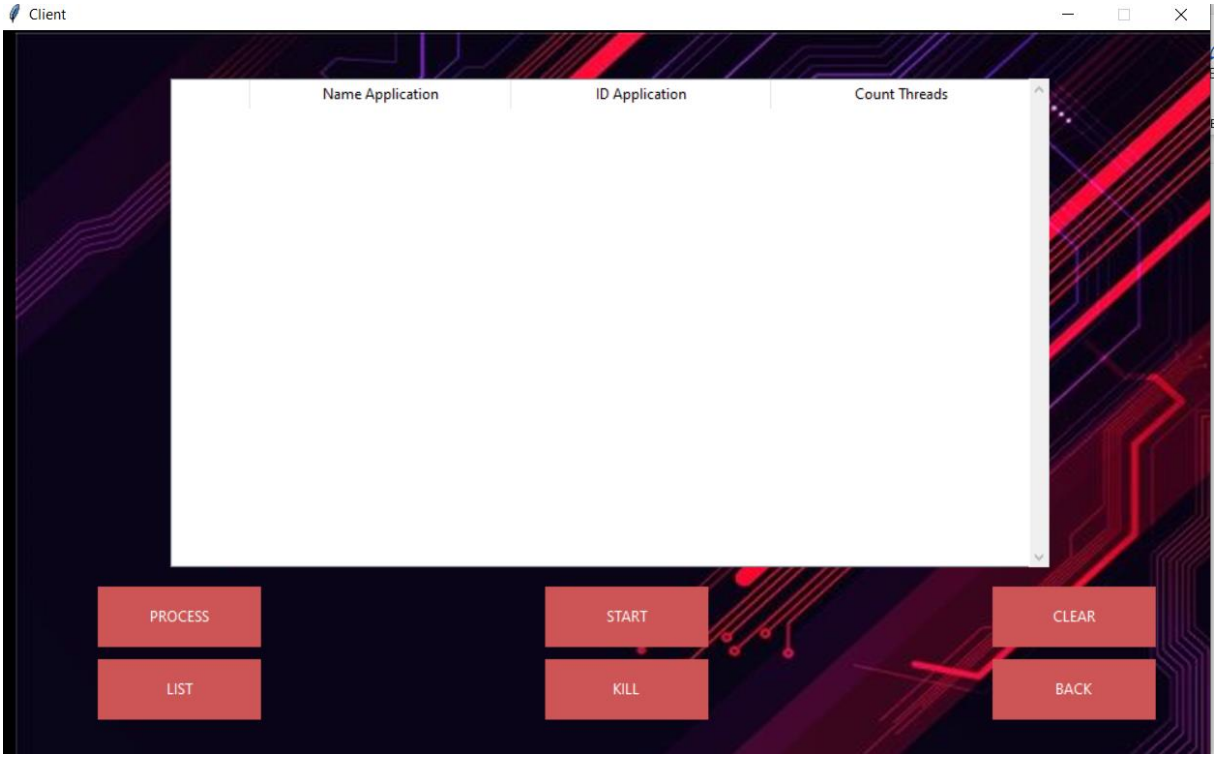
Giao diện của chức năng live screen:



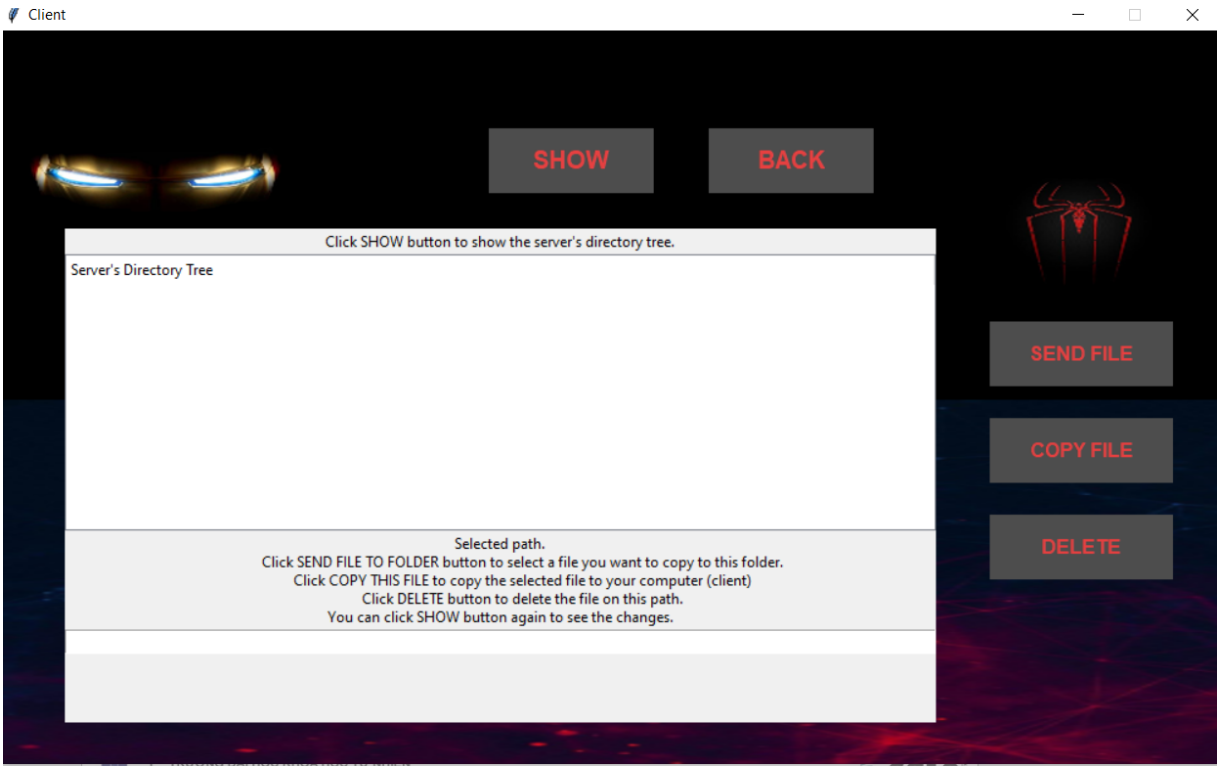
Giao diện chức năng keylogger



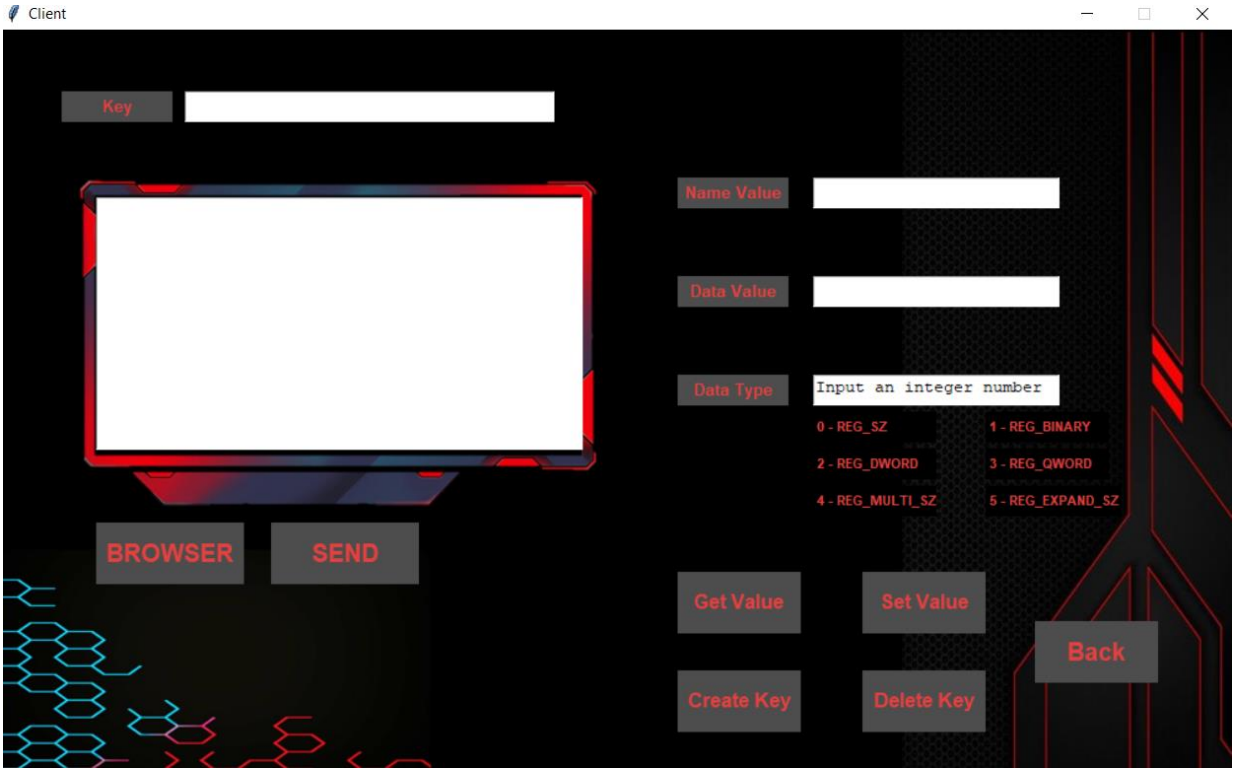
Giao diện chức năng app process:



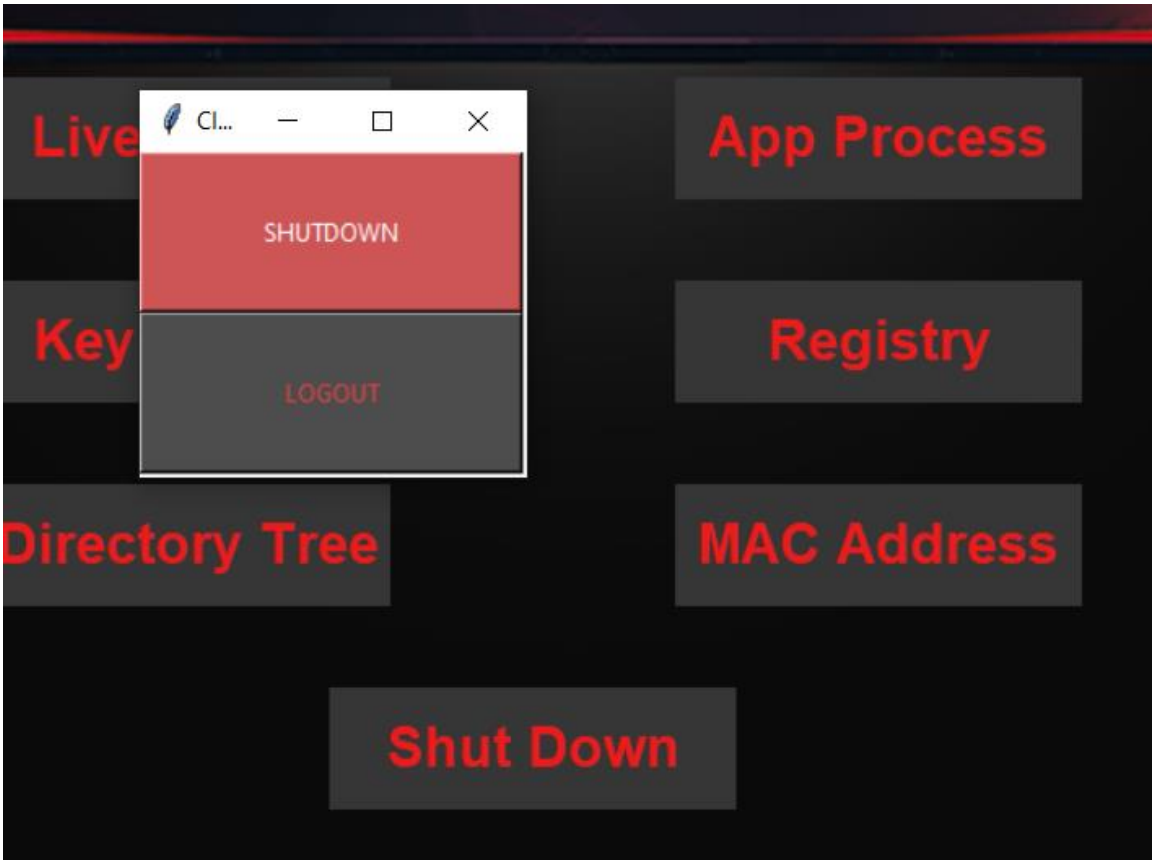
Giao diện chức directory tree:



Giao diện chức năng registry:



Giao diện chức năng của shut down + MAC address:



IV.Chức năng và mô tả các hàm:

Các thư viện chính sử dụng trong chương trình:

- Thư viện tkinter: dùng để xử lý giao diện người dùng
- Thư viện PIL: dùng để xử lý hình ảnh
- Thư viện os: cung cấp các chức năng được sử dụng để tương tác với hệ điều hành và cũng có được thông tin liên quan về nó
- Thư viện sys: cung cấp các hàm và các biến được sử dụng để thao tác các phần khác nhau của môi trường chạy Python. Nó cho phép chúng ta truy cập các tham số và chức năng cụ thể của hệ thống
- Thư viện re: thao tác với Regular Expression trong Python
- Thư viện winreg: truy cập vào window registry access
- Thư viện json: giúp thao tác json một định dạng dữ liệu rất phổ biến, được dùng để lưu trữ và thể hiện các dữ liệu có cấu trúc
- Thư viện threading: xử lý đa luồng trong python
- Thư viện pynput: cho phép bạn kiểm soát và giám sát các thiết bị đầu vào

- Thư viện keyboard: sử dụng để có toàn quyền kiểm soát bàn phím
- Thư viện pickle: triển khai các giao thức nhị phân để tuần tự hóa và hủy tuần tự hóa cấu trúc đối tượng Python
- Thư viện psutil: là một thư viện đa nền tảng để truy xuất thông tin về các quy trình đang chạy và sử dụng hệ thống (CPU, bộ nhớ, đĩa, mạng, cảm biến) bằng Python
- Thư viện struct: thực hiện chuyển đổi giữa các giá trị Python và cấu trúc C được biểu diễn dưới dạng đối tượng byte Python
- Thư viện subprocess: phép bạn tạo ra các quy trình mới, kết nối với các đường input/output/error pipes, and lấy mã trả lại của chúng
- Thư viện io: cung cấp các phương tiện chính của Python để xử lý các loại I / O khác nhau

Hàm lấy đường dẫn file hình ảnh mà các file thường hay sử dụng:

❖ Hàm path(file_name):

```
def path(file_name):
    file_name = 'pic\\' + file_name
    """ Get absolute path to resource, works for dev and for PyInstaller """
    try:
        # PyInstaller creates a temp folder and stores path in _MEIPASS
        base_path = sys._MEIPASS
    except Exception:
        base_path = os.path.abspath(".")

    return os.path.join(base_path, file_name)
```

- Hàm này người dùng có thể truyền vô một tên file bất kì,
- File_name sẽ tạo ra một chuỗi đường dẫn gồm 1 folder + 1 file hình ảnh chứa trong folder đó
- Sử dụng try except đê kiểm tra trong quá trình lấy đường dẫn file hiện tại python
- base_path = sys._MEIPASS, nếu trong quá trình lấy có xảy ra lỗi sẽ trả lại một đường dẫn tuyệt đối “.”
- sys._MEIGPASS, sẽ lại dường dẫn file tới folder chứa code python hiện tại
- Trả về một đường dẫn tổng giữa nơi chứa file code python + file_name mà mình đã tạo bằng os.path.join()

1) Giao diện đăng nhập và giao diện trang điều khiển:

File: logInPage_GUI.py

Thư viện cần có: tkinter, PIL, os, sys

Giao diện người dùng đăng nhập được viết bởi OOP với frame

- Tạo ra một đối tượng Login_Page_UI và được kế thừa từ Frame(trong thư viện tkinter), tạo hàm khởi tạo với self, parent.
- Trong hàm def __init__(self, parent) hàm khởi tạo
- Frame.__init__(self, parent) khởi tạo lớp cha
- self.configure() khởi tạo một frane với màu nên là bg = "#ff0000" chiều dài là 500, chiều rộng là 800, relief = "ridge"
- kích thước cửa số được làm là parent.geometry("800x500+200+200") rộng 800 dài 500 tọa độ xuất hiện trên màn hình pc là x=200 y=200
- self.grid(row=0, column=0, sticky="nsew") để sắp sếp một frame

File: homePage_GUI.py

Thư viện cần có: tkinter, os, sys

Giao diện hiện lên tất cả các chức cung cấp cho người dùng được viết bởi OOP với frame

Tương tự bên logInPage_GUI

Nhưng có sự thay đổi kích thước màn thành rộng 1000 chiều dài 600

Trên giao diện này t sẽ thấy các nút bấm……..

2) Chức năng live screen:

a. Kịch bản sử dụng:

- Khi người dùng bấm vào nút live screen ở giao diện chính thì chương trình sẽ chuyển qua giao diện của chức năng live screen gồm hình ảnh của màn hình hiện tại bên máy sever và phía dưới là 2 nút Back và Save.Và hình ảnh đang hiển thị trên màn hình sẽ luôn được cập nhật liên tục nhờ vào quá trình chụp nhanh và gửi nhận dữ liệu qua lại vô cùng nhanh chóng từ cả 2 phía sever và client.
- Khi người dùng bấm nút Save thì sẽ hiện ra 1 cửa sổ để người dùng đặt tên file ảnh muốn lưu và chọn nơi để lưu ảnh dưới dạng format png.
- Khi người dùng bấm nút Back thì chương trình sẽ đưa người dùng quay trở lại giao diện chính

b. Giao diện chức năng:

- Giao diện chức năng live screen được viết bởi OOP với frame
- Tạo một đối tượng Desktop_UI kế thừa từ Frame và khởi tạo bởi parent và một biên client giữ socket đang kết nối server, kích thước rọng 1000 dài 600 tọa độ xuất hiện x=200, y=200

```
# label to display frames received from server
self.label = tk.Label(self)
self.label.place(x=20,y=0,width=960,height=540)
```

- Sử dụng label để nhìn thấy hình ảnh xuất hiện trên màn hình giao diện
- Với 2 button sẽ xuất hiện trên màn hình là Back và Save

```
# thread
self.start = Thread(target=self.ChangeImage, daemon=True)
self.start.start()
```

- Sử dụng thread để liên tục cập nhật hình ảnh lên GUI

c. Mô tả chi tiết các hàm và thủ túc của chức năng:

File: live_screen_client.py

Thư viện cần có: threading, PIL, io, tkinter

❖ Hàm ChangeImage

```
# display frames continously
def ChangeImage(self):
    while self.status:
        size = int(self.client.recv(100))
        #self.client.sendall(bytes("READY", "utf8"))

        data = b""
        while len(data) < size:
            packet = self.client.recv(999999)
            data += packet

        image_PIL = Image.open(io.BytesIO(data)).resize((960, 540), Image.ANTIALIAS)
        image_tk = ImageTk.PhotoImage(image_PIL)
        self.label.configure(image=image_tk)
        self.label.image = image_tk

        # check save image command
        # while saving image, server will delay capturing and wait for the next command from
        if self.on_save:
            self.frame = data
            self.save_img()
            self.on_save = False

        # check stop command
        if self.status:
            self.client.sendall(bytes("NEXT_FRAME", "utf8"))
        else:
            self.client.sendall(bytes("STOP_RECEIVING", "utf8"))
    # Return the main UI
    self.destroy()
```

- Nhiệm vụ của hàm này là sẽ nhận liên tục các hình ảnh được chụp từ client và hiển thị liên tục lên màn hình cho đến khi status = false thì dừng nhận.

- Biến sz sẽ chứa thông tin về size của bức ảnh còn dữ liệu kiểu byte của bức ảnh sẽ được lưu trong biến data.
- Vì dữ liệu hình ảnh khi được gửi qua đã được chuyển đổi dưới dạng nhị phân và để có thể hiện thị đầy đủ trên cửa sổ giao diện thì ta phải resize nó nên ta phải sử dụng dụng thư viện io.BytesIO để thao tác với kiểu dữ liệu nhị phân và thư viện Image để resize lại bức ảnh với filter ANTIALIAS.
- Sau khi bức ảnh được điều chỉnh xong thì sẽ được lưu vào biến image_tk
- Và cuối cùng ta sẽ chèn image_tk và label để có thể hiển thị lên giao diện người dùng
- Nếu biến on_save bằng true thì ta sẽ gán dữ liệu trong biến data và biến frame và gọi hàm save_img và chỉnh lại giá trị biến on_save = false.
- Sau khi lưu ảnh thì ta sẽ check lại biến status để xem có nên gửi lệnh dừng nhận dữ liệu tới sever không.
- Sau khi kết thúc vòng while lớn thì ta sẽ gọi hàm place_forget để quay trở về giao diện chính

❖ Hàm click_back

```
def click_back(self):
    self.status = False
```

- Hàm này có nhiệm vụ là điều chỉnh biến status thành false để ngừng quá trình nhận ảnh diễn ra trong while lớn của hàm ChangeImage và sau đó sẽ trở về giao diện chính khi kết thúc dòng while lớn đó.

❖ Hàm click_save

```
def click_save(self):
    self.on_save = True
```

- Là hàm dùng để thay đổi biến on_save thành true để quá trình save ảnh trong hàm ChangeImage được diễn ra.

❖ Hàm save_img

```
def save_img(self):
    if self.frame == None:
        return

    types = [('Portable Network Graphics', '*.png'), ('All Files', '*.*')]
    image_file = asksaveasfile(mode='wb', filetypes=types, defaultextension='*.png')
    if image_file == None:
        return
    image_file.write(self.frame)
```

- Hàm này có nhiệm vụ là chọn nơi để lưu hình ảnh dưới dạng file png theo ý người dùng bằng cách sử dụng hàm có sẵn asksaveasfile
- Và nếu không có dữ liệu trong biến frame hoặc không tìm được chỗ để lưu thì quá trình lưu sẽ kết thúc hàm bằng lệnh return.

File: live_screen_server.py

Thư viện cần có: PIL, io

❖ Hàm capture_screen()

```
def capture_screen(client):
    INFO_SIZE = 100
    while client:
        image = ImageGrab.grab()
        image_bytes = io.BytesIO()
        image.save(image_bytes, format='PNG')
        data = image_bytes.getvalue()

        # send frame size
        client.sendall(bytes(str(len(data)), "utf8"))

        # send frame data
        client.sendall(data)

        # listen to next command from client: continue or back
        check_stop = client.recv(INFO_SIZE).decode("utf8")
        if("STOP_RECEIVING" in check_stop):
            break
```

- Biến image sẽ chứa hình ảnh được chụp nhanh bằng ImageGrab.grab()
- Biến image_byte sẽ là 1 buffer trong quá trình lưu ảnh dưới dạng nhị phân
- Biến data dùng để chứa byte của hình ảnh đang được lưu trong buffer image_byte
- Sau đó ta sẽ lần lượt gửi frame size và frame data đến cho client qua biến socket client.
- Quá trình chụp ảnh và gửi ảnh sẽ liên tục được diễn ra nhờ có vòng lặp while và chỉ đến biến check_stop chứa lệnh nhận từ client là “STOP_RECEIVING” thì mới kết thúc.

3) Chức năng key logger

a. Kịch bản sử dụng:

- Khi người dùng bấm vào nút keylogger ở giao diện chính thì chương trình sẽ chuyển sang giao diện của tính năng keylogger gồm 1 textbox để hiển thị danh sách các key đã bắt được và các nút chức năng:HOOK,PRINT,DELETE,LOCK và BACK.
- Khi người dùng bấm vào nút HOOK thì chương trình sẽ bắt đầu bắt phím từ sever và nút hook sẽ được chuyển thành UNHOOK để khi bấm vào sẽ dừng quá trình bắt phím từ sever.
- Khi người dùng bấm vào PRINT thì chương trình sẽ in ra trên màn hình chuỗi các phím đã bắt được,nếu trước đó chưa bắt được phím nào thì sẽ không có gì hiện lên màn hình
- Khi người dùng bấm vào nút DELETE thì chương trính sẽ xóa chuỗi các phím đã bắt được đang hiển thị trên màn hình
- Khi người dùng bấm vào nút LOCK thì chương trình sẽ khóa tất cả các phím bên sever đồng thời nút LOCK sẽ chuyển thành nút UNLOCK để khi bấm vào sẽ tắt khóa phím bên sever
- Khi người dùng nhấn vào nút BACK thì chương trình sẽ quay trở về giao diện chính của ứng dụng

b. Giao diện chức năng:

- Giao diện chức năng key logger được viết theo OOP với frame
- Tạo một đối tượng Keylogger_UI kế thừa từ Frame và khởi tạo bởi parent và một biên client giữ socket đang kết nối server, kích thước rọng 1000 dài 600 tọa độ xuất hiện x=200, y=200

```
self.text_1 = Text(
    self, height = 200, width = 500, state = "disable", wrap = "char",
    bd=0,
    bg='white',
    highlightthickness=0
)
self.text_1.place(
    x=220,
    y=100,
    width=600,
    height=360
)
```

- Khi màn hình giao diện xuất hiện, xuất hiện một hôp text với mục đích cho biết bên server gửi thông tin của keybooard bên server.
- Bên cạnh đó xuất các nút button như HOOK, LOCK, PRINT, DELETE, BACK ứng với mỗi chức năng mà chương trình cung cấp.

c. Mô tả chi tiết các hàm và thủ túc của chức năng:

File: Keylogger_client.py

Thư viện cần có: tkinter, PIL

❖ Hàm hook

```
def hook(client, button):
    client.sendall(bytes("HOOK", "utf8"))
    if button['text'] == "HOOK":
        button.configure(text = "UNHOOK")
    else:
        button.configure(text = "HOOK")
    return
```

- Hàm này có nhiệm vụ là gửi lệnh HOOK qua biến socket client và điều chỉnh thông điệp hiển thị của nút button.Nếu thành phần text trong button là HOOK thì ta sẽ chỉnh lại text thành UNHOOK và ngược lại ta sẽ chỉnh lại text thành HOOK

❖ Hàm _print

```
def _print(client, textbox):
    client.sendall(bytes("PRINT", "utf8"))
    print("  ")
    data = client.recv(BUFSIZ).decode("utf8")
    #data = data.replace("'", "")
    data = data[1:]
    textbox.config(state = "normal")
    textbox.insert(tk.END, data)
    textbox.config(state = "disable")
    return
```

- Hàm này có nhiệm vụ là gửi lệnh PRINT và nhận dữ liệu trả về qua biến socket client.
- Dữ liệu trả về sẽ được lưu vào biến data và ta sẽ chèn data và textbox để hiển thị dữ liệu
- Nhưng để chèn được được dữ liệu và textbox thì phải đặt trạng thái của textbox là normal và đặt trạng thái của textbox là disable sau khi chèn để tránh việc dữ liệu hiển thị trên textbox có thể bị điều chỉnh.

❖ Hàm delete

```
def delete(textbox):
    textbox.config(state = "normal")
    textbox.delete("1.0", "end")
    textbox.config(state = "disable")
    return
```

- Hàm này có nhiệm vụ là xóa dữ liệu đang hiển thị trên textbox và để có thể xóa được thì ta phải chỉnh trạng thái của textbox là normal và chỉnh lại trạng thái thành disable để tránh việc dữ liệu hiển thị trên textbox bị điều chỉnh.

❖ Hàm lock

```
def lock(client, button):
    client.sendall(bytes("LOCK", "utf8"))
    if button['text'] == "LOCK":
        button.configure(text = "UNLOCK")
    else:
        button.configure(text = "LOCK")
    return
```

- Hàm này có nhiệm vụ là gửi lệnh LOCK qua biến socket client và chỉnh lại thông điệp hiển thị của nút button.Nếu thành phần text trong button là LOCK thì ta sẽ chỉnh lại text thành UNLOCK và ngược lại ta sẽ chỉnh lại text thành LOCK

❖ Hàm back

```
def back():
    return
```

- Hàm này có nhiệm vụ thực hiện chức năng quay trở lại giao diện chính từ giao diện của keylogger

File: Keylogger_server.py

Thư viện cần có: threading, pynput, keyboard

❖ Hàm keylog

```
def keylog(client):
    global cont, flag, islock, ishook
    islock = 0
    ishook = 0
    threading.Thread(target = listen).start()
    flag = 0
    cont = " "
    message = ""
    while True:
        message = client.recv(BUFSIZ).decode("utf8")
        if "HOOK" in message:
            if ishook == 0:
                flag = 1
                ishook = 1
            else:
                flag = 2
                ishook = 0
        elif "PRINT" in message:
            _print(client)
        elif "LOCK" in message:
            lock()
        elif "QUIT" in message:
            flag = 4
            return
    return
```

- Hàm keylog có nhiệm vụ là điều khiển hoạt động của các hàm trong file keylogger_sever_py
- client là 1 biến socket để nhận lệnh từ client.
- message là biến chứa lệnh nhận từ client.
- cont là biến global chứa chuỗi các phím đã nghe lén được
- islock là biến điều khiển hoạt động của chức năng khóa bàn phím
- ishook và flag là biến điều khiển hoạt động của chức năng bắt các phím nhập từ bàn phím
- Đầu tiên ta để bắt đầu các thao tác với thread ta sẽ dùng lệnh start() và ta sẽ truyền từ khóa listen vào mục target để gọi hàm listen()

```
threading.Thread(target = listen).start()
```

- Khi biến client nhận dữ liệu từ client với BUFSIZ = 1024 * 4 và giãi mã bằng format utf8 thì tin nhắn sẽ được lưu trữ trong biến msg.
- Nếu lệnh trong biến message là "HOOK" thì ta sẽ kiểm tra giá trị của biến ishook.Nếu biến ishook có giá trị bằng 0 nghĩa là ta chưa bắt đầu bắt phím thì sẽ set giá trị của biến flag = 1 và ishook = 1 và ngược lại thì sẽ set giá trị flag = 2 và ishook = 0.Để hiểu thêm về tác dụng của việt set giá trị của biến flag,vui xem qua phần giải thích hoạt động của hàm keylogger
- Khi lệnh trong biến message là "PRINT",ta sẽ gọi hàm _print và truyền vào tham số là client
- Khi lệnh trong biến msg là "LOCK" ta sẽ gọi hàm lock()
- Khi lệnh trong biến msg là "QUIT" ta sẽ set giá trị flag = 4 và kết thúc hàm bằng lệnh return

❖ Hàm listen

```
def listen():
    with Listener(on_press = keylogger) as listener:
        listener.join()
    return
```

- Hàm listen có nhiệm vụ là bắt đầu nghe các phím mà người dùng đã nhập,ta sẽ truyền vào mục on_press từ khóa keylogger để gọi hàm keylogger.
- listener là 1 thread nên để kết nối với main thread thì ta sẽ dùng lệnh join()

❖ Hàm keylogger

```
def keylogger(key):
    global cont, flag
    if flag == 4:
        return False
    if flag == 1:
        temp = str(key)
        if temp == 'Key.space':
            temp = ' '
        elif temp == '"\'"':
            temp = "'"
        else:
            temp = temp.replace("'", "")
        cont += str(temp)
    return
```

- Hàm keylogger có chức năng ghi lại các phím đã nhập từ bàn phím.Khi giá trị flag = 1 thì biến temp sẽ lưu các phím đã nhập từ bàn phím dưới dạng chuỗi và chuỗi chứa trong biến temp sẽ được ghép vào biến global cont.Bên cạnh các ký tự như chữ cái thì t cũng phải xử lý các ký tự đặc biệt để quá trình lưu vào chuỗi không bị đứt quãng và chính xác như khi gặp ký tự "/".
- Bởi vì hàm keylogger được gọi bởi thread listener nên hàm keylogger sẽ luôn được gọi lại liên tục để lưu lại các phím nhập từ bàn phím và quá trình gọi lại đó chỉ kết thúc khi ta return false khi giá trị biến flag = 4.Đó chính là lý do ta thay đổi giá trị biến global flag = 4 khi nhận lệnh "QUIT" trong hàm keylog.
- Những giá trị ngoài 1 và 4 chỉ làm cho việc lưu ký tự nhập từ bàn phím vào chuỗi bị tạm dừng và hàm keylogger vẫn bị gọi lại liên tục.

❖ Hàm _print

```
def _print(client):
    global cont
    client.sendall(bytes(cont, "utf8"))
    cont = " "
    return
```

- Hàm _print có nhiệm vụ gửi lại client chuỗi các phím đã bắt được lưu trong biến global cont.Sau khi gửi thì ta sẽ set lại biến cont thành 1 chuỗi rỗng

❖ Hàm lock

```
def lock():
    global islock
    if islock == 0:
        for i in range(150):
            keyboard.block_key(i)
        islock = 1
    else:
        for i in range(150):
            keyboard.unblock_key(i)
        islock = 0
    return
```

- Hàm lock có nhiệm vụ là khóa bàn phím.Nếu giá trị biến global islock = 0 thì ta sẽ tạo 1 vòng for và sử hàm block_key của thư viện keyboard để khóa phím và cập nhật lại giá trị của biến islock = 1.Và ngược lại ta sẽ sử dụng hàm unblock_key của thư viện keyboard để bỏ khóa phím và cập nhật lại giá trị biến islock = 0.

4) Chức năng Directory tree:

a. Kịch bản sử dụng:

- Người dùng bấm vào nút Directory tree giao diện chính thì chương trình sẽ tự sang giao diện của chức năng directory tree, chúng ta sẽ thấy treeview
- Người dùng có thể nhìn thấy những lời hướng dẫn như "Click SHOW button to show the server's directory tree" ," Selected path
  Click SEND FILE TO FOLDER button to select a file you want to copy to this folder
  Click COPY THIS FILE to copy the selected file to your computer (client)
  Click DELETE button to delete the file on this path.\nYou can click SHOW button again to see the changes"
- Khi người dùng bấm nút SHOW để xem server directory tree
- Khi người dùng bấm SEND FILE gửi một file từ client cho server
- Khi người dùng bấm COPY FILE gửi một file từ server cho client
- Khi người dùng bấm DELETE xóa những thông tin hiện trên treeview
- Khi người dùng bấm Back để trở về giao diện

b. Giao diện chức năng:

- Giao diện chức năng Directory tree được viết theo OOP với frame
- Tạo một đối tượng Registry_UI kế thừa từ Frame và khởi tạo bởi parent và một biên client giữ socket đang kết nối server, kích thước rọng 1000 dài 600 tọa độ xuất hiện x=200, y=200

```
self.frame = tk.Frame(self, height = 200, width = 500)
self.tree = ttk.Treeview(self.frame)

self.frame.place(
    x=53,
    y=162,
    width=713,
    height=404
)

self.insText1 = "Click SHOW button to show the server's directory tree."
self.label1 = tk.Label(self.frame, text=self.insText1)
self.label1.pack(fill = tk.X)

ysb = ttk.Scrollbar(self.frame, orient='vertical', command=self.tree.yview)
xsb = ttk.Scrollbar(self.frame, orient='horizontal', command=self.tree.xview)
self.tree.configure(yscroll=ysb.set, xscroll=xsb.set)
self.tree.heading('#0', text='Server\'s Directory Tree', anchor='w')
self.tree.pack(fill = tk.BOTH)

self.tree.bind('<<TreeviewOpen>>', self.open_node)
self.tree.bind("<<TreeviewSelect>>", self.select_node)
```

- Sử dụng ttk.Treeview() để hiện thông tin directory tree

c. Mô tả chi tiết hàm và thủ tục của chức năng:

File directory_tree_client.py

Thư viện cần có: os, tkinter, pickle, PIL

❖ Hàm showTree()

```
def showTree(self):
    self.deleteTree()
    self.client.sendall("SHOW".encode())

    data_size = int(self.client.recv(BUFFER_SIZE))
    self.client.sendall("received filesize".encode())
    data = b""
    while len(data) < data_size:
        packet = self.client.recv(999999)
        data += packet
    loaded_list = pickle.loads(data)

    for path in loaded_list:
        try:
            abspath = os.path.abspath(path)
            self.insert_node('', abspath, abspath, True)
        except:
            continue
```

```
def deleteTree(self):
    self.currPath = " "
    self.path.config(state = "normal")
    self.path.delete("1.0", tk.END)
    self.path.config(state = "disable")
    for i in self.tree.get_children():
        self.tree.delete(i)
```

➔ Xoá cây đang hiển thị bằng phương thức delete()

- **Client** sau khi gửi lệnh "SHOW" -> nhận về size của filename và kiểm tra -> nhận dữ liệu và insert_node vào tree (thêm các file/folder vào tree)

❖ **Hàm insert_node()** -> them node vào tree bằng phương thức tree.insert()

```
def insert_node(self, parent, text, abspath, isFolder):
    node = self.tree.insert(parent, 'end', text=text, open=False)
    if abspath != "" and isFolder:
        self.nodes[node] = abspath
        self.tree.insert(node, 'end')
```

- Nếu người dùng click vào 1folder để mở ra xem các fiolder/file con

❖ **hàm (open_node):**

```
def open_node(self, event):
    node = self.tree.focus()
    abspath = self.nodes.pop(node, None)
    if abspath:
        self.tree.delete(self.tree.get_children(node))
        try:
            dirs = listDirs(self.client, abspath)
            for p in dirs:
                self.insert_node(node, p[0], os.path.join(abspath, p[0]), p[1])
        except:
            messagebox.showerror(message = "Cannot open this directory!")
```

-> lấy danh sách = listDirs và insert_node các file/folder con

- client sẽ lấy danh sách file/folder con bằng **phương thức listDirs():**

```
def listDirs(client, path):
    client.sendall(path.encode())

    data_size = int(client.recv(BUFFER_SIZE))
    if (data_size == -1):
        messagebox.showerror(message = "Click SHOW button again to watch the new directory tree!")
        return []
    client.sendall("received filesize".encode())
    data = b""
    while len(data) < data_size:
        packet = client.recv(999999)
        data += packet
    if (data == "error"):
        messagebox.showerror(message = "Cannot open this directory!")
        return []

    loaded_list = pickle.loads(data)
    return loaded_list
```

- Phương thức listDirs gửi cho server đường dẫn mà người dung click vào sau đó nhận về data là directory con (folder/file con).
- lưu data bằng pickle và return data vừa lưu.

❖ COPYTO:

```
# copy file from client to server
def copyFileToServer(self):
    self.client.sendall("COPYTO".encode())
    isOk = self.client.recv(BUFFER_SIZE).decode()
    if (isOk == "OK"):
        filename = filedialog.askopenfilename(title="Select File",
                                              filetypes=[("All Files", "*.*")])
        if filename == None or filename == "":
            self.client.sendall("-1".encode())
            temp = self.client.recv(BUFFER_SIZE)
            return
        destPath = self.currPath + "\\"
        filesize = os.path.getsize(filename)
        self.client.send(f"{filename}{SEPARATOR}{filesize}{SEPARATOR}{destPath}".encode())
        isReceived = self.client.recv(BUFFER_SIZE).decode()
        if (isReceived == "received filename"):
            try:
                with open(filename, "rb") as f:
                    data = f.read()
                    self.client.sendall(data)
            except:
                self.client.sendall("-1".encode())
            isReceivedContent = self.client.recv(BUFFER_SIZE).decode()
            if (isReceivedContent == "received content"):
                messagebox.showinfo(message = "Copy successfully!")
                return True
    messagebox.showerror(message = "Cannot copy!")
    return False
```

- Client gửi lệnh “COPYTO” (copy từ client qua Server)
- Client gửi filename, filesize, và path qua server để copy

-> nếu server ‘‘received filename’’

-> đọc file (filename) = read()

->gửi data vừa đọc cho server và xử lí

-

❖ COPY (copy từ Server qua client)

(client)

```
# copy file from server to client
def copyFileToClient(self):
    self.client.sendall("COPY".encode())
    isOk = self.client.recv(BUFFER_SIZE).decode()
    if (isOk == "OK"):
        try:
            destPath = filedialog.askdirectory()
            if destPath == None or destPath == "":
                self.client.sendall("-1".encode())
                temp = self.client.recv(BUFFER_SIZE)
                return
            self.client.sendall(self.currPath.encode())
            filename = os.path.basename(self.currPath)
            filesize = int(self.client.recv(100))
            if (filesize == -1):
                messagebox.showerror(message = "Cannot copy!")
                return
            self.client.sendall("received filesize".encode())
            data = b""
            while len(data) < filesize:
                packet = self.client.recv(999999)
                data += packet
            with open(destPath + "\\" + filename, "wb") as f:
                f.write(data)
            messagebox.showinfo(message = "Copy successfully!")
        except:
            messagebox.showerror(message = "Cannot copy!")
    else:
        messagebox.showerror(message = "Cannot copy!")
```

- Client chọn nơi lưu file copy bằng filedialog

-> sau đó gửi tên file cần copy cho server

->server nhận được và gửi data của file cho client như trên

->Client nhận được data

-> và tạo file cần copy ở đường dẫn đã chọn bằng filedialog.

❖ DEL :

```
def deleteFile(self):
    self.client.sendall("DEL".encode())
    isOk = self.client.recv(BUFFER_SIZE).decode()
    if (isOk == "OK"):
        self.client.sendall(self.currPath.encode())
        res = self.client.recv(BUFFER_SIZE).decode()
        if (res == "ok"):
            messagebox.showinfo(message = "Delete successfully!")
        else:
            messagebox.showerror(message = "Cannot delete!")
    else:
        messagebox.showerror(message = "Cannot delete!")
```

- Client nhận “ok”

->remove thành công

- hoặc “error”

-> remove không thành công

➔ gửi messagebox thông báo lên màn hình.

❖ QUIT:

- Server nhận lệnh "QUIT" từ ***client*** và return để dừng ***directory tree*** và thoát ra màn hình client

```
elif (mod == "QUIT"):
    return

else:
    client.sendall("-1".encode())
```

****

- Nếu người dung click chọn 1 file/folder trong directory tree -> thực thi hàm select_node()

```
def select_node(self, event):
    item = self.tree.selection()[0]
    parent = self.tree.parent(item)
    self.currPath = self.tree.item(item,"text")
    while parent:
        self.currPath = os.path.join(self.tree.item(parent)['text'], self.currPath)
        item = parent
        parent = self.tree.parent(item)

    self.path.config(state = "normal")
    self.path.delete("1.0", tk.END)
    self.path.insert(tk.END, self.currPath)
    self.path.config(state = "disable")
```

File directory_tree_server.py

❖ HÀM directory()

Thư viện cần có: os, pickle

```
def directory(client):
    isMod = False

    while True:
        if not isMod:
            mod = client.recv(BUFSIZ).decode()

        if (mod == "SHOW"):
            showTree(client)
            while True:
                check = sendListDirs(client)
                if not check[0]:
                    mod = check[1]
                    if (mod != "error"):
                        isMod = True
                        break
        # copy file from client to server
        elif (mod == "COPYTO"):
            client.sendall("OK".encode())
            copyFileToServer(client)
            isMod = False
        # copy file from server to client
        elif (mod == "COPY"):
            client.sendall("OK".encode())
            copyFileToClient(client)
            isMod = False
        elif (mod == "DEL"):
            client.sendall("OK".encode())
            delFile(client)
            isMod = False
        elif (mod == "QUIT"):
            return
        else:
            client.sendall("-1".encode())
```

- mod là biến lưu lệnh nhận được từ client với mod = SHOW / COPYTO / COPY / DEL / QUIT.

❖ **SHOW** -> thực thi **phương thức showTree(client):**

```
def showTree(sever):
    ListDirectoryTree = []
    for c in range(ord('A'), ord('Z') + 1):
        path = chr(c) + ":\\"
        if os.path.isdir(path):
            ListDirectoryTree.append(path)
    data = pickle.dumps(ListDirectoryTree)
    sever.sendall(str(len(data)).encode())
    temp = sever.recv(BUFSIZ)
    sever.sendall(data)
```

➔ Duyệt tên ổ đĩa và lưu vào listDirectoryTree nếu tồn tại
➔ gửi size của data và data cho client.
➔ Client nhận được và lưu lại và thêm vào directory tree

Nếu người dung mở 1 node trong cây (mở 1 folder trong directory tree)

❖ Thực thi hàm sendListDirs()

```
def sendListDirs(sever):
    path = sever.recv(BUFSIZ).decode()
    if not os.path.isdir(path):
        return [False, path]

    try:
        listTree = []
        ListDirectoryTree = os.listdir(path)
        for d in ListDirectoryTree:
            listTree.append((d, os.path.isdir(path + "\\" + d)))

        data = pickle.dumps(listTree)
        sever.sendall(str(len(data)).encode())
        temp = sever.recv(BUFSIZ)
        sever.sendall(data)
        return [True, path]
    except:
        sever.sendall("error".encode())
        return [False, "error"]
```

- Server nhận đường dẫn:
-> kiểm tra nếu tồn tại
-> trả về data là list các folder/file con cho client :
(append vào listTree và lưu = pickle)

❖ **COPYTO:**

- Server sau khi nhận lệnh COPYTO (copy file from client to server)
-> thực thi phương thức copyFileToServer(client):

```
# copy file from client to server
def copyFileToServer(sever):
    received = sever.recv(BUFSIZ).decode()
    if (received == "-1"):
        sever.sendall("-1".encode())
        return
    filename, filesize, path = received.split(SEPARATOR)
    filename = os.path.basename(filename)
    filesize = int(filesize)
    sever.sendall("received filename".encode())
    data = b""
    while len(data) < filesize:
        packet = sever.recv(999999)
        data += packet
    if (data == "-1"):
        sever.sendall("-1".encode())
        return
    try:
        with open(path + filename, "wb") as file:
            file.write(data)
        sever.sendall("received content".encode())
    except:
        sever.sendall("-1".encode())
```

- Biến received lưu size của data được client gửi qua gồm **[filename, filesize, path] (**nhận được từ client) .
- Sau khi gửi "received filename" cho client, server nhận về data của file, từ đó tạo ra 1 file có tên giống như ở client và gán data mà client đã gửi qua.

❖ COPY (copy từ server qua client)

- **SERVER :** nếu nhận được lệnh COPY thì gửi ''OK'' cho client và thực thi phương thức copyFileToClient() :

```
# copy file from server to client
def copyFileToClient(sever):
    filename = sever.recv(BUFSIZ).decode()
    if filename == "-1" or not os.path.isfile(filename):
        sever.sendall("-1".encode())
        return
    filesize = os.path.getsize(filename)
    sever.sendall(str(filesize).encode())
    temp = sever.recv(BUFSIZ)
    with open(filename, "rb") as f:
        data = f.read()
        sever.sendall(data)
```

- Server nhận được filename từ client
-> kiểm tra filename có hợp lệ hay không
->gửi size của filename cho client để kiểm tra
-> đọc file = read()
-> gửi data lại cho client.

❖ **DEL (delete file)**

```
elif (mod == "DEL"):
    client.sendall("OK".encode())
    deleteFile(client)
    isMod = False
```

- Server nhận được lệnh "DEL" thì thực thi phương thức **deleteFile() :**

```
def deleteFile(sever):
    file_name = sever.recv(BUFSIZ).decode()
    if os.path.exists(file_name):
        try:
            os.remove(file_name)
            sever.sendall("ok".encode())
        except:
            sever.sendall("error".encode())
            return
    else:
        sever.sendall("error".encode())
        return
```

- Server nhận đường dẫn file cần delete

->sau đó kiểm tra và xoá bằng phương thức os.remove()

-> gửi lại lệnh "ok" nếu remove thành công

-> gửi lại lệnh "error" nếu remove không thành công

❖ **QUIT**

- Server nhận lệnh "QUIT" từ ***client*** và return để dừng ***directory tree*** và thoát ra màn hình client

```
elif (mod == "QUIT"):
    return

else:
    client.sendall("-1".encode())
```

5) Chức năng App Process:

a. Kịch bản sử dụng:

- Khi người dùng bấm nút App Process ở giao diện chính thì chương trình sẽ tự chuyển sang giao diện của chức năng app process ở đây chúng sẽ một treeview và 6 nút bấm lần lượt là process, list, start, kill, clear, back
- Khi người dùng bấm người dùng bấm process, sau đó bấm list thì treeview hiện lên những ứng dụng chạy ngầm có trong server lúc này nút process chuyển thành application

- Người dùng muốn coi những ứng dụng nào đang chạy trên server bấm application sau đó bấm list
- Khi người dùng muốn khởi động một ưng dụng bấm start, sẽ xuất hiện một cửa sổ giao diện người dùng nhập tên ứng dụng và bấm nút start
- Khi người dùng muốn kill một ứng dụng, tương tự với start, bấm vào kill và nhập ID của ứng dụng xuất hiện trên treeview
- Muốn cập nhật lại thông tin treeview bấm clear sau đó bấm list
- Trở về giao diện chính bấm back

b. Giao diện chức năng:

- Giao diện chức năng App Process được viết theo OOP với frame
- Tạo một đối tượng Registry_UI kế thừa từ Frame và khởi tạo bởi parent và một biên client giữ socket đang kết nối server, kích thước rọng 1000 dài 600 tọa độ xuất hiện x=200, y=200

```
# dung tree view
self.tab = ttk.Treeview(self, height = 18, selectmode='browse')
self.scroll = tk.Scrollbar(self, orient = "vertical", command = self.tab.yview)
self.scroll.place(
    x=850,
    y=40,
    height=404
)
self.tab.configure(yscrollcommand = self.scroll.set)
self.tab['columns'] = ("Name", "ID", "Count")
self.tab.column('#0', width=0)
self.tab.column("Name", anchor="center", width = 150, minwidth = 10, stretch = True)
self.tab.column("ID", anchor="center", width = 150, minwidth = 10, stretch = True)
self.tab.column("Count", anchor="center", width = 150, minwidth = 10, stretch = True)
self.tab.heading('#0', text='')
self.tab.heading("Name", text = "Name Application")
self.tab.heading("ID", text = "ID Application")
self.tab.heading("Count", text = "Count Threads")
self.tab.place(
    x=140,
    y=40,
    width=713,
    height=404
)
```

- Sử dụng ttk.Treeview() để hiện thông tin directory tree

c. Mô tả chi tiết hàm và thủ tục của chức năng:
File app_process_client.py

Thư viện cần có: PIL, tkinter, pickle, struct

❖ Hàm send_kill()

```
def send_kill(client):
    global pid
    client.sendall(bytes("0", "utf8"))
    client.sendall(bytes(str(pid.get()), "utf8"))
    message = client.recv(BUFSIZ).decode("utf8")
    if "1" in message:
        tk.messagebox.showinfo(message = "Đã diệt!")
    else:
        tk.messagebox.showerror(message = "Lỗi!")
    return
```

- Hàm này có nhiệm vụ là gửi lệnh **"0" (kill app/process) + id của app/process** đó qua biến socket client và nhận về kết quả có kill thành công hay không thông qua biến res.

❖ Hàm _list()

```
def _list(client, tab, s):
    client.sendall(bytes("1", "utf8"))
    client.sendall(bytes(s, "utf8"))
    list1 = receive(client)
    list1 = pickle.loads(list1)
    list2 = receive(client)
    list2 = pickle.loads(list2)
    list3 = receive(client)
    list3 = pickle.loads(list3)
    print(list1)
    print(list2)
    print(list3)
    for i in tab.get_children():
        tab.delete(i)
    for i in range(len(list1)):
        tab.insert(parent = '', index = 'end', text = '', values = (list1[i], list2[i], list3[i]))
    return
```

- Sau khi lấy được list name, id, thread thông qua **ls1, ls2, ls3 bằng**

❖ Hàm receive() thì in ra màn hình.

```
def recvall(sock, size):
    message = bytearray()
    while len(message) < size:
        buffer = sock.recv(size - len(message))
        if not buffer:
            raise EOFError('Could not receive all expected data!')
        message.extend(buffer)
    return bytes(message)

def receive(client):
    packed = recvall(client, struct.calcsize('!I'))
    size = struct.unpack('!I', packed)[0]
    data = recvall(client, size)
    return data
```

❖ Hàm send_start

```
def send_start(client):
    global pname
    client.sendall(bytes("3", "utf8"))
    client.sendall(bytes(str(pname.get()), "utf8"))
    return
```

- Gửi lệnh "3" (start 1 app/process) và tên của app/process đó..

File app_process_server.py

Thư viện cần có: pickle, psutil, struct, os, subprocess

❖ HÀM app_process

```
def app_process(client):
    global msg
    while True:
        msg = client.recv(BUFSIZ).decode("utf8")
        if "QUIT" in msg and len(msg) < 20:
            return
        result = 0
        list1 = list()
        list2 = list()
        list3 = list()
        option = int(msg)
        #0-kill
        if option == 0:
            pid = client.recv(BUFSIZ).decode("utf8")
            pid = int(pid)
            try:
                result = kill(pid)
            except:
                result = 0
        #1-xem
        elif option == 1:
            try:
                status = client.recv(BUFSIZ).decode("utf8")
                if "PROCESS" in status:
                    list1, list2, list3 = list_apps()
                else:
                    list1, list2, list3 = list_processes()
                result = 1
            except:
                result = 0
```

```
    #3 - start
    elif option == 3:
        program_name = client.recv(BUFSIZ).decode("utf8")
        try:
            start(program_name)
            result = 1
        except:
            result = 0
    if option != 1 and option != 3:
        client.sendall(bytes(str(result), "utf8"))
    if option == 1:
        list1 = pickle.dumps(list1)
        list2 = pickle.dumps(list2)
        list3 = pickle.dumps(list3)

        send_data(client, list1)
        send_data(client, list2)
        send_data(client, list3)
return
```

- Hàm app_process có nhiệm vụ là điều khiển hoạt động của các hàm trong file app_process_sever_py
- client là 1 biến socket để nhận lệnh từ client.
- msg là biến chứa lệnh nhận từ client.

- Khi biến client nhận dữ liệu từ client với BUFSIZ = 1024 * 4 và giải mã bằng format utf8 thì tin nhắn sẽ được lưu trữ trong biến msg.
- Nếu lệnh trong biến msg là "QUIT" thì kết thúc hàm bằng return

- Biến action lưu msg dưới dạng int nếu không phải là QUIT theo quy ước: 0 – kill app/process, 1 - xem , 3 – start.
- action = 0(kill) -> gọi phương thức kill(pid) và trả về 0 cho biến res nếu không kill được và ngược lại.

```
#0-kill
if option == 0:
    pid = client.recv(BUFSIZ).decode("utf8")
    pid = int(pid)
    try:
        result = kill(pid)
    except:
        result = 0
```

- action = 1(xem) -> biến status nhận lệnh từ client để chọn xem list apps hay list processes. Sau đó gọi hàm list_apps hoặc list_processes.

```
elif option == 1:
    try:
        status = client.recv(BUFSIZ).decode("utf8")
        if "PROCESS" in status:
            list1, list2, list3 = list_apps()
        else:
            list1, list2, list3 = list_processes()
        result = 1
    except:
        result = 0
```

- action = 3 (start 1 process) -> biến pnam lưu tên app/process muốn start được nhập từ người dung -> phương thức start(pname).

```
#3 - start
elif option == 3:
    program_name = client.recv(BUFSIZ).decode("utf8")
    try:
        start(program_name)
        result = 1
    except:
        result = 0
```

❖ **Hàm kill()**

```
def kill(pid):
    cmd = 'taskkill.exe /F /PID ' + str(pid)
    try:
        a = os.system(cmd)
        if a == 0:
            return 1
        else:
            return 0
    except:
        return 0
```

- Os.system(cmd) -> thực thi command "cmd". (command kill app/process).

❖ Hàm list_apps()

```
def list_apps():
    list1 = list()
    list2 = list()
    list3 = list()

    cmd = 'powershell "gps | where {$_.mainWindowTitle} | select Description, ID, @{Name=\'ThreadCount\';Expression ={$_.Threads
    proc = os.popen(cmd).read().split('\n')
    temp = list()
    for line in proc:
        if not line.isspace():
            temp.append(line)
    temp = temp[3:]
    for line in temp:
        try:
            arr = line.split(" ")
            if len(arr) < 3:
                continue
            if arr[0] == '' or arr[0] == ' ':
                continue

            name = arr[0]
            threads = arr[-1]
            ID = 0
            # interation
            cur = len(arr) - 2
            for i in range (cur, -1, -1):
                if len(arr[i]) != 0:
                    ID = arr[i]
                    cur = i
                    break
            for i in range (1, cur, 1):
                if len(arr[i]) != 0:
                    name += ' ' + arr[i]
            list1.append(name)
            list2.append(ID)
            list3.append(threads)
        except:
```

- Chạy command lấy list apps và lưu vào biến **proc.**
- đưa dữ liệu từng app vào biến tmp (list), và dễ dàng lấy được tên, id,… của app thông qua việc split(' ').
- -> lưu list name, id, threads vào ls1, ls2, ls3 và return.

❖ Hàm list_process()

```
def list_processes():
    list1 = list()
    list2 = list()
    list3 = list()
    for proc in psutil.process_iter():
        try:
            # Get process name & pid from process object.
            name = proc.name()
            pid = proc.pid
            threads = proc.num_threads()
            list1.append(str(name))
            list2.append(str(pid))
            list3.append(str(threads))
        except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
            pass
    return list1, list2, list3
```

- Dễ dàng thấy được thông tin các process đang chạy thông qua phương thức **psutil.process_iter()** và lưu vào ls1, ls2, ls3.

❖ **Hàm start**

```
def start(name):
    subprocess.Popen(name)
    return
```

- Dễ dàng start 1 app/process bằng module os với phương thức os.system().
- Nếu action = 1 (xem app/process) thì cần gửi dữ liệu list app/process về cho client:

6) Chức năng Registry:

a. Kịch bản sử dụng:

- Người dùng bấm vào nút Registry ở giao diện chính thì chương trình sẽ tự chuyển sang giao diện của chức năng registry chúng ta sẽ thấy 4 hộp text dùng để nhập key, Name value, Data Value, Data Type, dùng để nhập lần lượt các thông tin.
- Một text box lớn để hiện nội dung file tải từ máy tính client lên xem nội dung.
- Khi người dùng bấm nút BROWSER dùng để tải một file có đuôi *.reg lên và xem nội dung
- Khi người dùng bấm nút SEND gửi nội dung file vừa tải được cho server, ngoài ra người dùng có thể tự ghi nội dung mình muốn chỉnh sữa ở registry lên text box và bấm nút gửi.
- Khi người dùng nút Get Value trước hết phải nhập đường dẫn Key và Name Value
- Khi người dùng nút Set Value thì người dùng cần nhập đầy đủ thông tin
- Khi người dùng Create Key, Delete Key cần phải nhập đường dẫn key mà mình muốn tạo cũng như mình muốn xóa

- Bấm nút Back để trở về giao diện điều khiển.

b. Giao diện chức năng:

- Giao diện chức năng Registry được viết theo OOP với frame
- Tạo một đối tượng Registry_UI kế thừa từ Frame và khởi tạo bởi parent và một biên client giữ socket đang kết nối server, kích thước rọng 1000 dài 600 tọa độ xuất hiện x=200, y=200

```
# copy socket connection to own attribute
self.client = client
# attributes of registry keys/values
self.action_ID = None
self.key = None
self.name = None
self.data = None
self.data_type = 1
# attributes for response of action
self.res1 = None
self.res2 = None
# initialize status ready to use
self.status = True
```

- Ở phần thuộc tính của lớp ta khai báo có các thuộc tính như
- self.client = client , self.action_ID = None ,self.key = None, self.name = None, self.data = None, self.data_type = 1, self.res1 = None, self.res2 = None, self.status = True là các thuộc tính tin nhắn để làm việc với server

c. Mô tả chi tiết các hàm và thủ túc của chức năng:

File: registry_ client.py

Thư viện cần có: json, tkinter, os, sys, PIL

❖ Hàm get_input()

```
def get_input(self):
    self.key = self.key_txt.get("1.0", "end").rstrip()
    self.name = self.name_txt.get("1.0", "end").rstrip()
    self.data = self.data_txt.get("1.0", "end").rstrip()
    dtypes = ['REG_SZ', 'REG_BINARY', 'REG_DWORD', 'REG_QWORD', 'REG_MULTI_SZ', 'REG_EXPAND_SZ']
    try:
        t_num = int(self.data_type_txt.get("1.0", "end").rstrip())
        if(0 <= t_num and t_num <= 5):
            self.data_type = dtypes[t_num]
        else:
            self.data_type = dtypes[0]
    except:
        self.data_type = dtypes[0]
```

- Lấy dữ nhập vào từ GUI và gán lần lượt cho self.key, self.name, self.data bằng hàm get("1.0", "end").rstrip() trả về một chuỗi có trong hộp text và kí tự cuối
- Lưu danh sách các kiểu dữ liệu của value trong một list dtypes
- Để biết người dùng muốn kiểu dữ liệu nào, sử dụng hàm self.data_type_txt. get("1.0", "end").rstrip() là chuỗi kí tự là số, sau đó ta ép kiểu về kiểu int để biết người dùng muốn kiểu dữ liệu nào
- Nếu không Set kiểu dữ liệu băng một cái list của python nào sẽ hiểu tự động là 'REG_SZ'

❖ Hàm get_input()

```
def get_value(self):
    self.get_input()
    self.action_ID = 1
    self.send_msg()
```

- Hàm lấy gửi tin nhắn để lấy value từ registry server:
- Dùng hàm self.get_input() để nhập tất cả dữ liệu từ GUI
- Gán self.action_ID = 1 có ý nghĩa client sẽ gửi yêu cầu get value đã tồn tại cho server biết
- Đóng gói thông tin và thực hiện trao đổi giữa client và server bằng send_msg()
- Hàm để tạo một value mới ở và gửi đến registry server

❖ Hàm set_value()

```
def set_value(self):
    self.get_input()
    self.action_ID = 2
    self.send_msg()
```

- Dùng hàm self.get_input() để nhập tất cả dữ liệu từ GUI
- Gán self.action_ID = 2 có ý nghĩa client sẽ gửi yêu cầu set một value mới cho server thực hiện
- Đóng gói thông tin và thực hiện trao đổi giữa client và server bằng send_msg()
- Ham dùng để gửi tin nhắn cho server biết tạo một value mới

❖ Hàm create_key()

```
def create_key(self):
    self.get_input()
    self.action_ID = 3
    self.send_msg()
```

- Dùng hàm self.get_input() để nhập tất cả dữ liệu từ GUI
- Gán self.action_ID = 3 có ý nghĩa client sẽ gửi yêu cầu tạo một key mới cho server thực hiện
- Đóng gói thông tin và thực hiện trao đổi giữa client và server bằng send_msg()
- Hàm để gửi yêu cầu tạo một Key đến server

❖ Hàm Delete_key()

```
def delete_key(self):
    self.get_input()
    self.action_ID = 4
    self.send_msg()
```

- Dùng hàm self.get_input() để nhập tất cả dữ liệu từ GUI
- Gán self.action_ID = 4 có ý nghĩa client sẽ gửi yêu cầu delete một key đã tồn tại cho server thực hiện
- Đóng gói thông tin và thực hiện trao đổi giữa client và server bằng send_msg()
- Hàm để gửi yêu cầu xóa một Key đến server

❖ Hàm send_detail()

```
def send_detail(self):
    self.get_input()
    s = self.content.get("1.0", "end")
    self.key = s
    self.action_ID = 0
    self.send_msg()
```

- Dùng hàm self.get_input() để nhập tất cả dữ liệu từ GUI
- Lấy tất cả dữ liệu có trong hộp text lớn GUI bằng hàm self.content.get("1.0", "end") sau đó gán self.key = s một chuỗi vừa đọc được
- Gán self.action_ID = 0 có ý nghĩa client sẽ gửi yêu một file *.reg cho server sau đó server sẽ thực hiện cập nhật lại Registry bằng nội dung có trong file.

- Đóng gói thông tin và thực hiện trao đổi giữa client và server bằng send_msg()
- Hàm để gửi một file *.reg đến server

❖ Hàm open_file()

```
def open_file(self):
    file = filedialog.askopenfilename()
    if file == None or file == '':
        return
    self.content.delete("1.0", 'end')
    s=""
    with open(file, 'r') as input_file:
        s = input_file.read()
    for line in s:
        self.content.insert(tk.END, line)
```

- Hàm này để mở một file từ client
- File = filedialog.askopenfilename() để mở một đượng dẫn tới file mà ta muốn chọn
- Nếu lấy không được hàm sẽ lập tức thoat
- Thực hiện xóa tất cả các nội dung có trên hộp text bằng hàm self.content.delete(“1.0”, “end”)
- Tạo một chuỗi s để lưu thông tin mà có trong file mà ta vừa lấy
- Tiến hành đọc nội dung file vào biến s
- Ghi các nội dung vừa đọc được từ file lên hộp text với từng dòng bằng lệnh
- self.content.insert(tk.END, line)

❖ Hàm send_msg()

```
def send_msg(self):
    msg = {'ID' : self.action_ID, 'path' : self.key, 'name_value' : self.name, 'value' : self.data, 'v_type' : self.data_type}
    msg = json.dumps(msg)
    msg_bytes = bytes(msg, 'utf8')
    msg_sz = str(len(msg_bytes))
    self.client.sendall(bytes(msg_sz, 'utf8'))
    self.client.sendall(msg_bytes)
    self.res1 = self.client.recv(BUFSIZ).decode('utf8')
    self.res2 = self.client.recv(BUFSIZ).decode('utf8')
    if self.action_ID == 1:
        if '0' in self.res1:
            tk.messagebox.showerror(title='Thông báo', message='Thao tác không hợp lệ')
        else:
            tk.messagebox.showinfo(title='Thông báo', message=self.res2)
    else:
        if '0' in self.res1:
            tk.messagebox.showerror(title='Thông báo', message='Thao tác không hợp lệ')
        else:
            tk.messagebox.showinfo(title='Thông báo', message='Thành công')
```

- Ở hàm này sử dụng kiểu dữ liệu Dictionary của python với nhưng cặp key-value ứng với những yêu cầu, tên biến khác nhau
- ID: self.action_ID: lưu lại những hoạt động client muốn server sẽ làm
- Path: self.key: lưu lại đường dẫn key và hive
- Name_value: self.name: lưu lại tên value
- Value: self.data: lưu lại nội dung data của value
- V_type: self.data_type: lưu lại kiểu dữ liệu của value
- Tuần tự hóa đối tượng thành một chuỗi có định dạng JSON
- Mã hóa chuỗi dữ liệu có định dạng JSON băng hàm bytes(msg,'utf8') theo format = utf8
- Tạo một biến lưu kích thước của chuỗi sau khi mã hóa
- Client sẽ gửi 2 đoạn tin nhắn tới cho server gồm lần thứ nhất gửi thông tin về kích thước của dữ liệu sẽ gửi, đoạn tin nhắn thứ 2 client sẽ gửi cho server là dữ liệu đã được mã hóa
- Client sẽ tiếp ngồi chờ bên server gửi về một cái list chứa thông tin hành động của server thực hiện và một đoạn tin nhắn báo, hay dữ liệu đã thực hiện bên server
- Self.res1 = self.client.recv(BUFSIZ).decode(utf8) # nhan ve thông tin server đã làm
- Self.res2 = self.client.recv(BUFSIZ).decode(utf8) # ma doan tin nhan da thanh cong hay that bai
- Nếu là thao tác là get value client sẽ tiến hành kiểm tra thông tin của res1 đã nhận nếu băng 0 sẽ sai hành động và sẽ thông báo ra GUI băng messagebox
- Ngược lại sẽ thực hiện nếu đúng hành động đó

❖ Hàm click_back()

```
def click_back(self):
    self.status = False
    self.client.sendall(bytes("STOP_EDIT_REGISTRY", "utf8"))
    return
```

- Chuyển trạng thái của self.status = false cho biết clien muons thoát và không thực hiện trao đổi tiếp
- Gửi cho server biết stop edit registry

File: registry_ client.py

Thư viện cần có: re, winreg. Json, os

❖ Hàm parse_data(full_path)

```
def parse_data(full_path):
    try:
        full_path = re.sub(r'/', r'\\', full_path)
        hive = re.sub(r'\\.*$', '', full_path)
        if not hive:
            raise ValueError('Invalid \'full_path\' param.')
        if len(hive) <= 4:
            if hive == 'HKLM':
                hive = 'HKEY_LOCAL_MACHINE'
            elif hive == 'HKCU':
                hive = 'HKEY_CURRENT_USER'
            elif hive == 'HKCR':
                hive = 'HKEY_CLASSES_ROOT'
            elif hive == 'HKU':
                hive = 'HKEY_USERS'
        reg_key = re.sub(r'^[A-Z_]*\\', '', full_path)
        reg_key = re.sub(r'\\[^\\]+$', '', reg_key)
        reg_value = re.sub(r'^.*\\', '', full_path)
        return hive, reg_key, reg_value
    except:
        return None, None, None
```

- Tiến hành sẽ thay thế tất cả các kết quả khớp với pattern trong chuỗi bằng một nội dung khác được truyền vào và trả về chuỗi đã được sửa đổi hàm re.sub(r'\', r'\\', full_path) thay các kí tự đăt biệt thanh dấu '\\'
- Tiến hành lấy hive băng hàm re.sub(r'\\.*$','',full_path) trong đó r'\\.*$' có ý nghĩa là sẽ lấy đoạn chuỗi đầu tiên có kết thúc chuỗi là \\ với dấu ($) Biểu tượng dấu hoa thị (*) có thể khớp với chuỗi có hoặc không có ký tự được định nghĩa trước nó và dấu (.) khớp với bất kỳ ký tự đơn thông thường nào ngoại trừ ký tự tạo dòng mới \n
- Tiến hành kiểm tra việc lấy hive có thành công hay không, nếu sai sử dụng class ValueError() để
- Thông báo đường dẫn không hợp lệ
- Nếu hive của t là 4 kí tự thì tiến hành kiểm tra và trả về chuỗi hive đúng
- Tiến hành tách key co trong hive bằng lệnh
- reg_key = re.sub(r'^[A-Z_]*\\', '', full_path) (1)
- reg_key = re.sub(r'\\[^\\]+$', '', reg_key) (2)
- (1) Loại bỏ các chuỗi có kí tự in hoa trước dấu "\\", (2) tìm kiếm các pattern sau dấu'\\' và ở cuối không có dấu'\\' và xóa đi.
- reg_value = re.sub(r'^.*\\', '', full_path) (3)
- thay thê đoạn chuỗi có có "\\" sau đuôi băng ' ' để lấy value
- Hàm trả về một các hive, key, value đã phân tích từ full-path

❖ Hàm get_value(full_path)

```
def get_value(full_path):
    value_list = parse_data(full_path)
    try:
        opened_key = winreg.OpenKey(getattr(winreg, value_list[0]), value_list[1], 0, winreg.KEY_READ)
        value_of_value, value_type = winreg.QueryValueEx(opened_key, value_list[2])
        winreg.CloseKey(opened_key)
        return ["1", value_of_value]
    except:
        return ["0", "0"]
```

- Truyền vào là một đường dẫn tới key mà mình muốn lấy value
- Lưu các hive,key,value dưới dạng list python với tên biên là value_list
- Sử dụng try and except để kiểm tra quá trình kết nối với registry băng hàm
- OpenKey(key, sub_key, reserved=0, access=KEY_READ)
- Hàm getattr(winreg, value_list[0]) để trả về một thuộc tính của một đối tượng cụ thể ở đây là winreg.*
- Với * là hive tương ứng, value_list[1] là đường dẫn tới key chứa value
- Sau khi đã mở key ta dùng hàm winred.QueryValueEx() để truy xuất value có ở trong key và trả về bộ giá trị được chỉ định liên kết với khóa đã mở trước đó
- Winreg.ClosedKey() đóng lại kết kết nối key đã mở
- Hàm trả về một list gồm ID thao tác đã thực hiện và giá trị value vừa được truy cập tới

❖ Hàm dec_value(c)

```
def dec_value(c):
    c = c.upper()
    if ord('0') <= ord(c) and ord(c) <= ord('9'):
        return ord(c) - ord('0')
    if ord('A') <= ord(c) and ord(c) <= ord('F'):
        return ord(c) - ord('A') + 10
    return 0
```

- Hàm cho viết giá trị thập phân cơ số 10 của một kí tự in hoa
- Kí tự c được truyền vào hàm sẽ chuyển thành chữ in hóa và tiến hành kiểm tra nếu c la kí tự số thì trả về ord(c) – ord('0') với hàm ord() có nhiệm vụ trả lại điểm mã unicode cho mỗi kí tự
- Nếu là kí tượng in hoa sẽ tương tự

❖ Hàm str_to_bin(s):

```
def str_to_bin(s):
    res = b""
    for i in range(0, len(s), 2):
        a = dec_value(s[i])
        b = dec_value(s[i + 1])
        res += (a * 16 + b).to_bytes(1, byteorder='big')
    return res
```

- Hàm chuyển kiểu string sang binary
- Đối số truyền vào của hàm là mốt tring với lần lươt là 2 kí tự có trong chuỗi, sẽ tiến hành đổi ra giá trị thập phấn sau đó mã hóa 2 kí tự thành kiểu bytes với công thức (a*16+b).to_bytes(1, byteorder = ‘big’) với độ dài byte là 1 mã hóa theo kiểu big.

❖ Hàm str_to_dec(s)

```
def str_to_dec(s):
    s = s.upper()
    res = 0
    for i in range(0, len(s)):
        v = dec_value(s[i])
        res = res*16 + v
    return res
```

- Hàm chuyển kiểu string thành một kiểu int
- Đối số truyền vào của hàm là chuỗi string s đoạn chuỗi được in hoa mỗi kí tự, với mỗi kí tự s được chuyển thành kiểu nhị phân và cộng vào biến res theo công thức res = res*16 + v

❖ Hàm set_value(full_path, value, value_type)

```
def set_value(full_path, value, value_type):
    value_list = parse_data(full_path)
    try:
        winreg.CreateKey(getattr(winreg, value_list[0]), value_list[1])
        opened_key = winreg.OpenKey(getattr(winreg, value_list[0]), value_list[1], 0, winreg.KEY_WRITE)
        if 'REG_BINARY' in value_type:
            if len(value) % 2 == 1:
                value += '0'
            value = str_to_bin(value)
        if 'REG_DWORD' in value_type:
            if len(value) > 8:
                value = value[:8]
            value = str_to_dec(value)
        if 'REG_QWORD' in value_type:
            if len(value) > 16:
                value = value[:16]
            value = str_to_dec(value)

        winreg.SetValueEx(opened_key, value_list[2], 0, getattr(winreg, value_type), value)
        winreg.CloseKey(opened_key)
        return ["1", "1"]
    except:
        return ["0", "0"]
```

- Hàm set giá value cho Registry
- Đối số truyền của hàm là một đoạn full_path chứa chuỗi đường dẫn tới key mà mình muốn set value, value là tên của value, value_type: kiểu data mà mình muốn set.
- Sử dụng try và except để kiểm tra quá trình thực hiện có lỗi hay không nếu lỗi trả về một list[“0”,”0”] cho biết không thực hiện thành công
- Tạo một key mới nếu key đó chưa có bằng winreg.CreateKey(getattr(winreg, value_list[0]), value_lisr1[1]) nếu đã tồn tại chức năng này sẽ mở khóa hiện có.
- Mở một key băng lệnh winreg.OpenKey(getattr(winreg, value_list[0]), value_lisr1[1],0,winreg.KEY_WRITE) chứa năng này mở khóa để ghi thông tin.
- Ta chuyển hóa dữ liệu mà muốn chỉnh sữa cho phù hợp kiểu dữ liệu đã khai báo trước đó ở client
- Nếu kiểu dữ liệu là REG_BINARY và độ dài của data là lẻ thì data tiến hành cộng thêm một kí tự ‘0’
- Sau đó tiến hành chuyển value kiểu string thành bytes
- Nếu kiểu dữ liệu là REG_DWORD mà trong độ dài value > 8 ta chỉ cần lấy 8 kí tự là đủ, và chuyển thành hệ thập phân.
- Nếu kiểu dữ liệu là REG_QWORD mà trong độ dài value > 16 ta chỉ cần lấy 16 kí tự là đủ, và chuyển thành hệ thập phân.
- Tiến hành chỉnh sữa value hoặc tạo value mới bằng winreg.SetValueEx(opened_key, value_list[2], 0, getattr(winreg, value_type), value) với đối số là key vừa mở,

value_list[2] là tên value, value là thông tin về data của value, getattr(winreg, value_type): để xác định kiểu dữ liệu của value.

- Đóng kết nối với key vừa tạo bằng hàm ClosedKey()
- Trả về list thông báo đã hoàn thành xong và không có lỗi

❖ Hàm delete_value(full_path)

```
def delete_value(full_path):
    value_list = parse_data(full_path)
    try:
        opened_key = winreg.OpenKey(getattr(winreg, value_list[0]), value_list[1], 0, winreg.KEY_WRITE)
        winreg.DeleteValue(opened_key, value_list[2])
        winreg.CloseKey(opened_key)
        return ["1", "1"]
    except:
        return ["0", "0"]
```

- Hàm delete một value trong registry
- Mở một key băng lệnh winreg.OpenKey(getattr(winreg, value_list[0]), value_lisr1[1],0,winreg.KEY_WRITE) chứa năng này mở khóa để ghi thông tin.
- Winreg.DeleteValue(opened_key, value_list[2]) để mở một key sau đó tiến hành xóa value (value_list[2])
- Đóng khóa vừa mở winreg.ClosedKey()
- Trả về kiểu list để thông báo không xảy ra lỗi

❖ Hàm create_key(full_path)

```
def create_key(full_path):
    value_list = parse_data(full_path)
    try:
        winreg.CreateKey(getattr(winreg, value_list[0]), value_list[1] + r'\\' + value_list[2])
        return ["1", "1"]
    except:
        return ["0", "0"]
```

- Hàm tạo một key
- Phân tích đoạn chuỗi chứ path key thành một list lưu tại value_list
- Sử dụng try và except đêt kiểm tra quá trình có lỗi hay không nếu có lỗi trả về môi list = [“0”,”0”]
- Sử dụng hàm CreateKey(getattr(winregm value_list[0]), value_list[1] + r ‘\\’ + value_list[2]) để tạo hoặc mở một key với các đối số getattr(winregm value_list[0]): key, value_list[1] + r ‘\\’ + value_list[2]: subkey

❖ Hàm delete_key(full_path)

```
def delete_key(full_path):
    value_list = parse_data(full_path)
    try:
        winreg.DeleteKey(getattr(winreg, value_list[0]), value_list[1] + r'\\' + value_list[2])
        return ["1", "1"]
    except:
        return ["0", "0"]
```

- Phân tích đoạn chuỗi chứ path key thành một list lưu tại value_list
- Sử dụng try và except đêt kiểm tra quá trình có lỗi hay không nếu có lỗi trả về môi list = [“0”,”0”]
- Sử dụng hàm DeleteKey(getattr(winregm value_list[0]), value_list[1] + r ‘\\’ + value_list[2]) để xóa một key với các đối số getattr(winregm value_list[0]): key, value_list[1] + r ‘\\’ + value_list[2]: subkey

❖ Hàm registry(client)

```
def registry(client):
    BUFSIZ = 32768
    while True:
        header = client.recv(BUFSIZ).decode("utf8")
        if("STOP_EDIT_REGISTRY" in header):
            break
        data_sz = int(header)
        data = b""
        while len(data) < data_sz:
            packet = client.recv(BUFSIZ)
            data += packet

        msg = json.loads(data.decode('utf8'))
        # extract elements
        ID = msg['ID']
        full_path = msg['path']
        name_value = msg['name_value']
        value = msg['value']
        v_type = msg['v_type']
        res = ['0','0']

        print(ID)
        print(full_path)
        print(name_value)
        print(value)
        print(v_type)
```

- Được đặt trong vòng while(true) giúp server có thể làm các thao tác liên tiếp khi nhận thông tin từ client
- Server nhận một thông tin của client đồng tiên về kích thước của kiểu dữ liệu bytes
- Kiểm tra trong đoạn tin nhắn nếu “STOP_EDIT_REGISTRY” server kết thức phiên giao tiếp
- Gán data_sz = int(header) chuyển về kiểu int

- Sử dụng vòng lặp for để đọc hết dữ liệu mà client gửi tới lần 2 đến đủ kích thước dữ liệu.
- Sau khi nhận xong mã hóa đoạn tin nhắn thành dạng chuỗi có định dạng JSON
- Msg = json.loads(data.decode('utf8'))
- Phân tích lưu các value có trong file json thành các biến
- ID = msg['ID']→ cho biết server sẽ làm gì
- Full_path = msg['path']→ chứa path đường dẫn key
- name_value = msg['name_value']→ tên value
- value = msg['value']→ chứa data của value
- v_type = msg['v_type']→ kiểu dữ liệu của value
- res = ['0','0']→ thông tin ban đầu server chưa làm gì hết

```
if ID == 0:
    try:
        outout_file = os.getcwd() + '\\run.reg'
        with open(outout_file, 'w+') as f:
            f.write(full_path)
            f.close()
        os.system(r'regedit /s ' + os.getcwd() + '\\run.reg')
        res = ["1", "1"]
        print('file reg created')
    except:
        res = ["0", "0"]
        print('cannot create file reg')

elif ID == 1:
    res = get_value(full_path + r'\\' + name_value)

elif ID == 2:
    res = set_value(full_path + r'\\' + name_value, value, v_type)

elif ID == 3:
    res = create_key(full_path)

elif ID == 4:
    res = delete_key(full_path + r'\\')
client.sendall(bytes(res[0], "utf8"))
client.sendall(bytes(str(res[1]), "utf8"))
```

- Nếu ID = 0 cho biết server sẽ tạo một file run.reg để edit registry
- Trong đó outout_file = os.getcwd() + '\\run.reg' , tạo một đường file hiện tại chứa code python + tên file
- with open(outout_file, 'w+') as f:
- f.write(full_path)
- f.close()

- Ghi file nhị phân với thông chứa trong full_path, sau khi đã tạo xong file run.reg sử dụng câu lệnh
- os.system(r'regedit /s ' + os.getcwd() + '\\run.reg') để chạy file run.reg mới được tạo
- res = ["1", "1"]
- print('file reg created'), thông báo cho server biết đã làm xong về đống gói tin nhắn để trả lời lại client
- trong quá trình thực hiện xảy ra lỗi try và except sẽ bắt được và trả về res = ["0", "0"]
- print('cannot create file reg')
- Nếu ID = 1: gọi hàm get_value(full_path + r'\\' + name_value) đối số lúc này truyền vô là một đường dẫn tới key nối với chuỗi tên name_value
- Nếu ID = 2: gọi hàm set_value(full_path + r'\\' + name_value, value, v_type) đối số lúc này truyền vô là một đường dẫn tới key nối với chuỗi tên name_value, data của value(value), kiểu dữ liệu của value(v_type).
- Nếu ID = 3: gọi hàm create_key(full_path) đối sô là chứa thông tin của một đường dẫn key
- Nếu ID = 4: gọi hàm delete_key(full_path + r'\\') đối sô là chứa thông tin của một đường dẫn key
- Sau khi đã thực hiện các thao tác server gửi một đoạn thông tin của list[1] (res) cho biết đến với client có thành công hay không tiếp theo server gửi cho client một đoạn tin nhắn của list[2](res) thông tin tin nhắn của các chức năng get_value còn các chức năng còn lại mặc định là 1.
- Server tiếp ngồi đợi tin nhắn của client lặp lại vòng while(true) đến khi client gửi tin kết thức thúc giao tiếp

7) Chức năng MAC address:

a. Kịch bản sử dụng:

- Người dùng chỉ cần nhấn nút MAC address ở giao diện điều khiển

b. Giao diện chức năng:

Là một hộp messagebox hiện lên thông báo địa chỉ cho người dùng

c. Mô tả chi tiết các hàm và thủ túc của chức năng:

File mac_address_client.py

Thư viện cần có: tkinter

❖ Hàm mac_address(client)

```
def mac_address(client):
    try:
        result = client.recv(BUFFSIZE).decode("utf8")
        result = result[2:].upper()
        result = ':'.join(result[i:i + 2] for i in range(0, len(result), 2))
        tk.messagebox.showinfo(title='MAC Address', message=result)
    except:
        message = "Check the connection again"
        tk.messagebox.showerror(title='MAC Address', message=message)
```

- Người dùng truyền vào là một client socket
- Sử dụng try and except để kiểm tra trong quá trình nhận dữ liệu nếu sai ở giao diện ta nhìn thấy một messagebox thông báo lỗi cho người dùng biết
- Nếu đúng client sẽ bắt đầu decode theo "utf8" lưu trữ trong biến result
- Sử dụng vòng lặp for để chuyển một chuỗi hệ thập lục phăn và đưa ra màn hình bởi messagebox.

File mac_address_server.py

Thư viện cần có: uuid

❖ Hàm mac_address(client)

```
def mac_address(client):
    client.sendall(bytes(hex(uuid.getnode()), "utf8"))
    return
```

- Server dụng hàm getnode() để lấy địa chỉ máy mà chuyển thành mã hexa bằng hàm hex()

8) Chức năng ShutDown:

a. Kịch bản sử dụng:

- Khi người dùng bấm vào nút shutdown/logout trong giao diện chính thì sẽ hiện lên 1 cửa sổ gồm 2 nút SHUTDOWN và LOGOUT.
- Nếu người dùng bấm vào nút LOGOUT thì máy bên sever sẽ logout ngay lập tức
- Nếu người dùng bấm vào nút SHUTDOWN thì sau 15 giây máy bên sever sẽ shutdown.

b. Giao diện chức năng:

```
def shutdown_logout(client, root):
    window = tk.Toplevel(root)
    window.geometry("190x160")
    window.grab_set()
    window.protocol("WM_DELETE_WINDOW", lambda: close_event(window, client))
    shutdown_button = tk.Button(
        window, text = 'SHUTDOWN', width = 20, height = 2, fg = 'white', bg = 'IndianRed3',
        command = lambda: shutdown(client), padx = 20, pady = 20)
    shutdown_button.grid(row = 0, column = 0)
    logout_button = tk.Button(
        window, text = 'LOGOUT', width = 20, height = 2, fg = '#e64040', bg = '#4d4d4d',
        command = lambda: logout(client), padx = 20, pady = 20)
    logout_button.grid(row = 1, column = 0)
    window.mainloop()
```

- Giao diện xuất hiện 2 nút bấm là shut down với log out với một chức năng bật một cửa sổ lên băng tk.Toplevel(root)

c. Mô tả chi tiết các hàm và thủ tục của chức năng:

File: shutdown_logout_client.py

Thư viện cần có: tkinter

❖ Hàm close_event

```
def close_event(main, client):
    client.sendall(bytes("QUIT", "utf8"))
    main.destroy()
    return
```

- Hàm này có nhiệm vụ là gửi lệnh QUIT đến sever thông qua biến socket client đồng thời đóng cửa sổ chứa các nút shutdown/logout qua biến main bằng thư viện có sẵn destroy

❖ Hàm shutdown

```
def shutdown(client):
    client.sendall(bytes("SHUTDOWN", "utf8"))
```

- Hàm này có nhiệm vụ là gửi lệnh SHUTDOWN tới sever qua biến socket client

❖ Hàm logout

```
def logout(client):
    client.sendall(bytes("LOGOUT", "utf8"))
```

- Hàm này có nhiệm vụ gửi lệnh LOGOUT tới sever qua biến socket client

❖ Hàm shutdown_logout

```
def shutdown_logout(client, root):
    window = tk.Toplevel(root)
    window.geometry("190x160")
    window.grab_set()
    window.protocol("WM_DELETE_WINDOW", lambda: close_event(window, client))
    shutdown_button = tk.Button(
        window, text = 'SHUTDOWN', width = 20, height = 2, fg = 'white', bg = 'IndianRed3',
        command = lambda: shutdown(client), padx = 20, pady = 20)
    shutdown_button.grid(row = 0, column = 0)
    logout_button = tk.Button(
        window, text = 'LOGOUT', width = 20, height = 2, fg = '#e64040', bg = '#4d4d4d',
        command = lambda: logout(client), padx = 20, pady = 20)
    logout_button.grid(row = 1, column = 0)
    window.mainloop()
```

- Hàm này có nhiệm vụ hiển thị cửa sổ chứa các nút shutdown/logout bằng biến window

Biến shutdown_button a và logout_button lần lượt là hai biến đại diện sự hiển thị của hai nút shutdown và logout trong chương trình

File: shutdown_logout_server.py

Thư viện cần có: os

```
def shutdown_logout(client):
    while(True):
        message = client.recv(BUFFSIZE).decode("utf8")
        if "SHUTDOWN" in message:
            os.system('shutdown -s -t 15')
        elif "LOGOUT" in message:
            os.system('shutdown -l')
        else:
            return
    return
```

- Lệnh nhận từ client thông qua biến socket client sẽ được lưu vào biến message.
- Nếu thông điệp trong biến message là SHUTDOWN thì ta sẽ gọi chức năng shutdown của hệ điều hành qua thư viện os.
- -s đại diện cho chức năng shutdown và -t 15 là hẹn giờ 15s để bắt đầu shutdown máy
- Còn nếu thông điệp trong biến message là LOGOUT thì ta sẽ gọi chức năng logout của hệ điều hành qua thư viện os
- -l là đại diện cho chức năng logout và bởi vì ta không khai báo -t nên sau khi thực thi chương trình thì máy sẽ logout ngay lập tức

9) Client và Server:

a. Client:

File client.py

```
#global variables
BUFSIZ = 1024 * 4
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

- Tạo một socket tên là client với tham sô đầu tiên là socket.AF_INET:Ipv4, tham số thứ hai là socket type: cách thiết lập giao thức socket.SOCK_ STREAM: TCP

```
app = Tk()
app.geometry("1000x600")
app.configure(bg = "#FFFFFF")
app.title('Client')
app.resizable(False, False)
frame_lg = LogIn_Page_UI(app)
```

- Khởi tạo một cửa sổ window băng Tk() và cho kích thước là bằng hàm geometry(“1000x600”), màn hình window không thể thay đổi kích thước bằng hàm resizable(False, False), gọi frame chứa Frame Login page đã làm trước đó
- Frame_lg = LogIn_Page_UI(app)

```
def main():
    frame_lg.button_1.configure(command = connect)
    app.mainloop()

if __name__ == '__main__':
    main()
```

- Trong hàm main app.mainloop() cho phép lặp lại xuất hiện của giao diện, gọi thuộc tính button_1, để kết nối với server bằng hàm connect

```
def connect():
    global client
    ip = frame_lg.entry_1.get()
    try:
        client.connect((ip, 5656))
        messagebox.showinfo(message = "Connect successfully!")
        show_main_ui()
    except:
        messagebox.showerror(message = "Cannot connect!")
    return
```

- Tạo một biến global client nhằm client trở thành biến toàn cục
- Đọc địa chỉ IP đã nhập ở loggin page bằng cách gọi frame_lg.entry_1.get()
- Sử dụng try và except để kiểm tra quá tình kết nối nếu lỗi messagebox sẽ xuất hiện vào thông báo người dùng không thể kết nối
- Ngược lại messagebox thông báo cho người dùng biết kết nối thành công, connect() kết nối tới IP và PORT của server đang lắng nghe., gọi hàm show_main_ui() để gọi màn hình điều khiển

```
def show_main_ui():
    frame_lg.destroy()
    global frame_hp
    frame_hp = HomePage_UI(app)
    frame_hp.button_live_creen.configure(command = liveCreen)
    frame_hp.button_mac_address.configure(command = mac_address)
    frame_hp.button_disconnect.configure(command = disconnect)
    frame_hp.button_keylogger.configure(command = keylogger)
    frame_hp.button_AppProcess.configure(command = app_process)
    frame_hp.button_directoryTree.configure(command = directory_tree)
    frame_hp.button_registry.configure(command = registry)
    frame_hp.button_shut_down.configure(command = shutdown_logout)
    return
```

- Xóa frame loggin page đi và tạo một biến toàn cục frame_hp
- Gọi frame giao diện điều khiển và khởi tạo đường dẫn của các nút live screen, mac address, disconnect, keylogger, app process, directory tree, registry, shut down vào các hàm tương ứng để gọi các frame tương ứng với mỗi chức năng

```
def liveCreen():
    client.sendall(bytes("LIVESCREEN", "utf8"))
    temp = lsc.Desktop_UI(app, client)
    if temp.status == False:
        back(temp)
    return
```

- Bắt đầu chức năng liveCreen, client sẽ gửi một tin nhắn bắt đầu ”LIVESCREEN” dưới định dạng “utf8”, gọi frame giao diện live screen bằng lsc.Desktop_UI(app, client)
- Kiểm tra thuộc tính status của giao diện nếu bằng False gọi hàm back() để trở về giao diện điều khiển.

```
def keylogger():
    client.sendall(bytes("KEYLOG", "utf8"))
    temp = kl.Keylogger_UI(app, client)
    temp.button_back.configure(command = lambda: back(temp))
    return
```

- Bắt đầu chức năng key logger, client sẽ gửi tin nhắn tới server “KEYLOG” dưới định dạng “utf8”,gọi frame giao diện key logger bằng kl.keylogger_UI(app, client)
- Truy cập thuộc tích button_back để tạo một đường dẫn nút back tới hàm back() để trở về giao diện điều khiển chính.

```
def directory_tree():
    client.sendall(bytes("DIRECTORY", "utf8"))
    temp = dt.DirectoryTree_UI(app, client)
    temp.button_6.configure(command = lambda: back(temp))
    return
```

- Bắt đầu chức năng directory tree , client sẽ gửi tin nhắn tới server “DIRECTORY” dưới định dạng “utf8”,gọi frame giao diện key logger bằng dt.DirectoryTree_UI(app, client)
- Truy cập thuộc tích button_6 để tạo một đường dẫn nút back tới hàm back() để trở về giao diện điều khiển chính.

```
def app_process():
    client.sendall(bytes("APP_PRO", "utf8"))
    temp = ap.App_Process_UI(app, client)
    temp.button_back.configure(command = lambda: back(temp))
    return
```

- Bắt đầu chức năng app process , client sẽ gửi tin nhắn tới server “APP_PRO” dưới định dạng “utf8”,gọi frame giao diện key logger bằng dt.DirectoryTree_UI(app, client)
- Truy cập thuộc tích button_back để tạo một đường dẫn nút back tới hàm back() để trở về giao diện điều khiển chính.

```
def registry():
    client.sendall(bytes("REGISTRY", "utf8"))
    temp = rc.Registry_UI(app, client)
    temp.btn_back.configure(command=lambda: back_reg(temp))
    return
```

- Bắt đầu chức năng registry, client sẽ gửi tin nhắn tới server “REGISTRY” dưới định dạng “utf8”,gọi frame giao diện key logger bằng dt.DirectoryTree_UI(app, client)
- Truy cập thuộc tích btn_back để tạo một đường dẫn nút back tới hàm back_reg() để trở về giao diện điều khiển chính.

```
def back_reg(temp):
    temp.client.sendall(bytes("STOP_EDIT_REGISTRY", "utf8"))
    temp.destroy()
    frame_hp.tkraise()
```

- Gửi tin nhắn server rằng “STOP_EDIT_REGISTRY”, với định dạng “utf8” rồi xóa đó ta tiến hành xóa đi giao diện hiện tại và bật lại giao diện màn hình chính bằng lệnh tkraise()

```
def shutdown_logout():
    client.sendall(bytes("SD_LO", "utf8"))
    temp = sl.shutdown_logout(client, app)
    return
```

- Bắt đầu chức năng shut down logout, client sẽ gửi tin nhắn tới server “SD_LO” dưới định dạng “utf8”,gọi frame giao diện key logger bằng dt.shutdown_logout (app, client)

```
def back(temp):
    temp.destroy()
    frame_hp.tkraise()
    client.sendall(bytes("QUIT", "utf8"))
```

- Xóa đi giao diện hiện tại, bật lại giao diện điều khiển chính và gửi cho server biết kết thúc phiên giao tiếp (“QUIT”)

b. Server:

```
def Connect():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = ''
    port = 5656
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((host, port))
    s.listen(100)
    global client
    client, addr = s.accept()
```

- Tạo một socket tên là s với tham số đầu tiên là socket.AF_INET:Ipv4, tham số thứ hai là socket type: cách thiết lập giao thức socket.SOCK_ STREAM: TCP
- Port = 5656
- Socket tạo ra không thể sử dụng lại.
- s.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1) để tránh gặp lỗi.
- s.bind() được sử dụng để liên kết socket với địa chỉ. Trong TCP socket phân biệt nhau bởi cặp IP và port. Vì vậy, tham số truyền vào là một tuple của IP và port.
- s.listen() server bắt đầu lắng nghe kết nối trên địa chỉ được được liên kết trước đó. Tham số truyền vào với ý nghĩa chỉ chấp nhận số lượng kết nối nhất định.
- s.accept() chấp nhận kết nối đến.

```
while True:
    msg = client.recv(BUFSIZ).decode("utf8")
    if "KEYLOG" in msg:
        keylogger()
    elif "SD_LO" in msg:
        shutdown_logout()
    elif "LIVESCREEN" in msg:
        live_screen()
    elif "APP_PRO" in msg:
        app_process()
    elif "MAC" in msg:
        mac_address()
    elif "DIRECTORY" in msg:
        directory_tree()
    elif "REGISTRY" in msg:
        registry()
    elif "QUIT" in msg:
        client.close()
        s.close()
        return
```

- Server bước vào lặp while nhằm thực hiện các yêu cầu liên tiếp, server nhân tin nhắn từ client và giải mã ra và tiến hành kiểm tra thông tin tin nhắn bằng lệnh if và thực hiện đúng những thông điệp đã gửi bằng cách gọi các hàm tương ứng
- "KEYLOG" gọi keylogger()
- "SD_LO" gọi shutdown_logout()
- "LIVESCREEN" gọi live_screen()
- "APP_PRO" gọi app_address()
- "DIRECTORY" gọi directory_tree()
- "REGISTRY" gọi registry()
- "MAC" gọi mac_address()
- "QUIT" đóng kết nối close(), s.closed() dừng vòng lặp và thoát khỏi hàm.

V. Phân công và đánh giá:

1. Phân cộng:

Tên	MSSV	Nhiệm vụ	Đánh giá
Hoàng Hữu Minh An	20127102	Giao diện chương trình Tìm hiểu và viết giải thích registry Tìm hiểu và viết giải thích mac address Viết báo cáo	Tốt

Lê Phan Duy Tùng	20127661	Tìm hiểu và viết giải thích key logger, live screen, shut down, logout Quay video demo Viết báo cáo	Tốt
Thái Văn Thiên	20127631	Tìm hiểu và viết giải thích về app process, directory tree Hỗ trợ quay video demo Viết báo cáo	Tốt

2. Đánh giá:

Mức độ hoàn thành đồ án: 100%

Tên - MSSV	Tỷ lệ đóng góp
Hoàng Hữu Minh An - 20127102	40%
Lê Phan Duy Tùng - 20127661	30%
Thái Văn Thiên - 20127631	30%