

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN MÔN HỌC MẠNG MÁY TÍNH

ĐỀ TÀI: Điều khiển máy tính thông qua ứng dụng
Desktop

Giảng viên lý thuyết: Thầy Đỗ Hoàng Cường

Giảng viên thực hành: Cô Huỳnh Thuy Bảo Trân

Lớp: 22TNT1TN

Thành viên thực hiện:

- 22120025 – Nguyễn Long Bảo
- 22120044 – Nguyễn Cao Cường
- 22120049 – Tạ Chí Thành Danh

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 12 NĂM 2023

Mục lục

1	Giới thiệu	2
1.1	Tổng quan	2
1.2	Về Socket	2
1.3	Công cụ	2
1.3.1	wxWidgets	2
1.3.2	Asio	3
1.3.3	Windows API	3
1.4	Các link	3
1.4.1	Link video demo	3
1.4.2	Link mã nguồn	4
2	Tổ chức thư mục của chương trình	5
3	Cú pháp email	6
4	Gửi và nhận email	7
4.1	Nhận mail	7
4.2	Gửi mail	7
5	Các hàm xử lý yêu cầu	8
5.1	Phân tích câu lệnh	8
5.2	Module mail_handler	9
5.3	Process/Application	10
5.4	Chụp màn hình	12
5.5	Webcam	12
5.6	Keylogger	13
5.7	Registry	14
5.8	Directory	16
5.9	Shutdown/log out/restart	17
6	Demo	18
	Lời cảm ơn	19

Danh sách hình vẽ

1	Tổ chức thư mục ở server	5
2	Tổ chức thư mục ở client	5

Danh sách bảng

1 Giới thiệu

1.1 Tổng quan

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

- Kết nối đến máy Server qua địa chỉ IP: Cho phép thiết lập kết nối tới máy chủ thông qua việc nhập địa chỉ IP.
- Đăng nhập với vai trò admin hoặc user bình thường: Cung cấp tính năng đăng nhập với vai trò quản trị hoặc người dùng thông thường, đảm bảo quyền truy cập phù hợp.
- Ngắt kết nối với một địa chỉ IP: Cho phép ngắt kết nối tới máy tính có địa chỉ IP cụ thể trong cùng mạng LAN.
- Chia sẻ màn hình hiện tại: Hiện thị và chia sẻ màn hình hiện tại của máy tính Server.
- Gửi tín hiệu chuột đến máy bị điều khiển: Điều khiển các thao tác chuột từ xa đến máy tính được điều khiển.
- Gửi tín hiệu bàn phím đến máy bị điều khiển: Cho phép điều khiển bàn phím từ xa trừ các tổ hợp phím đặc biệt như Ctrl + Alt + Del.

1.2 Về Socket

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

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

1.3 Công cụ

1.3.1 wxWidgets

wxWidgets là một thư viện mạnh mẽ và linh hoạt cho phát triển giao diện người dùng đa nền tảng bằng ngôn ngữ lập trình C++. Trong đồ án này, wxWidgets được sử dụng để thiết kế giao diện cho ứng dụng, đồng thời hỗ trợ việc xử lý các sự kiện từ phía Client và Server trong quá trình gửi và nhận các thông tin qua socket.

Thư viện này cung cấp các thành phần giao diện đồ họa như cửa sổ, nút bấm, hộp thoại và menu, giúp tạo ra giao diện người dùng trực quan và dễ sử dụng. Đặc biệt,

wxWidgets cho phép triển khai ứng dụng trên nhiều hệ điều hành khác nhau một cách dễ dàng mà không cần phải viết lại mã nguồn.

Việc sử dụng wxWidgets giúp nhóm tập trung vào việc thiết kế giao diện và chức năng điều khiển từ xa mà không cần quá lo lắng về sự không tương thích giữa các nền tảng hệ điều hành. Thư viện cũng cung cấp tài liệu phong phú và có cộng đồng hỗ trợ đông đảo, giúp việc học và phát triển ứng dụng trở nên dễ dàng hơn.

1.3.2 Asio

Thư viện Asio là một thư viện mã nguồn mở mạnh mẽ cho ngôn ngữ lập trình C++, được sử dụng để thực hiện việc lập trình mạng và giao tiếp qua mạng một cách linh hoạt và hiệu quả.

Asio cung cấp các công cụ và lớp trừu tượng để tạo, quản lý và tương tác với các kết nối mạng, bao gồm cả việc xử lý socket, thiết lập kết nối, gửi và nhận dữ liệu trên mạng. Đặc biệt, nó hỗ trợ cả các giao thức đồng bộ (ví dụ như TCP) và không đồng bộ (ví dụ như UDP), cho phép việc truyền dữ liệu một cách đáng tin cậy và nhanh chóng.

Asio được thiết kế để linh hoạt và dễ sử dụng, với cách tiếp cận dựa trên bất đồng bộ (asynchronous), cho phép thực hiện các hoạt động mạng mà không cần chờ đợi kết quả trả về từ mỗi yêu cầu riêng biệt. Điều này làm cho việc xây dựng ứng dụng mạng với hiệu suất cao trở nên dễ dàng hơn, đặc biệt trong việc xử lý hàng loạt yêu cầu từ nhiều nguồn khác nhau.

Trong đồ án này, nhóm tập trung sử dụng chủ yếu với chức năng thiết lập kết nối TCP của nó. Sự linh hoạt và hiệu suất của Asio trong việc xử lý giao tiếp mạng bất đồng bộ đã hỗ trợ nhóm trong việc xây dựng và quản lý việc truyền dữ liệu đáng tin cậy giữa các thiết bị qua mạng.

1.3.3 Windows API

Windows API (Application Programming Interface) là bộ công cụ mạnh mẽ của hệ điều hành Windows, cung cấp chức năng và dịch vụ cần thiết cho việc tương tác ứng dụng Windows với hệ thống và phần cứng.

Trong quá trình phát triển ứng dụng Remote Desktop Control, nhóm đã tận dụng tính linh hoạt của Windows API để thực hiện các thao tác điều khiển từ xa. Chúng tôi đã sử dụng Windows API để gửi các yêu cầu tương ứng với các thao tác như nhấn phím, di chuyển con chuột, và thực hiện các thao tác click chuột trên máy tính được điều khiển.

Hơn nữa, thông qua Windows API, chúng tôi đã có thể truy xuất và lấy thông tin cơ bản của máy tính được kết nối, như địa chỉ IP và địa chỉ MAC, giúp quản lý và xác định máy tính từ xa một cách chính xác.

Bằng cách tích hợp các tính năng linh hoạt của Windows API vào ứng dụng Remote Desktop Control, chúng tôi đã tạo ra một trải nghiệm điều khiển từ xa hiệu quả và ổn định, mang lại sự linh hoạt và tiện ích cho người dùng.

1.4 Các link

1.4.1 Link video demo

Chèn link demo vào đây

1.4.2 Link mã nguồn

Còn đây là link mã nguồn

2 Tổ chức thư mục của chương trình

Chương trình được tổ chức như sau. Ở server:

```
.
├─ mail_parser.py
├─ mail_provider_handle
│   └─ POP3_service.py
│   └─ env.py
│   └─ SMTP_service.py
├─ requirements.txt
├─ mail_handler.py
├─ main.py
└─ old_handle
    ├─ mac_address_server.py
    ├─ app_process_server.py
    └─ shutdown_logout_server.py
```

Hình 1: Tổ chức thư mục ở server

Server có thư mục **mail_provider_handle** chứa các module phục vụ cho quá trình gửi và nhận mail, thư mục **old_handle** chứa một vài hàm được cung cấp sẵn cho quá trình xử lý các yêu cầu.

Module **mail_parser.py** chứa hàm phục vụ cho quá trình phân tích lệnh được nhận từ email. Module **mail_handler** chứa các hàm xử lý yêu cầu, giao tiếp với client và trả kết quả.

Tập tin **requirements.txt** chứa các yêu cầu cần thiết (bên cạnh Python 3.9 trở lên) để chạy chương trình.

Tập tin **main.py** là tập tin chính của server, ta cần chạy tập tin này để khởi động server. **Lưu ý** khi chạy tập tin này, ta cần thay đổi giá trị **MAX_CONNECTION** ở dòng 13 thành giá trị **bằng đúng** với số lượng client (tối đa 4). Ở client:

```
.
├─ app_process_client.py
├─ keylogger_client.py
├─ shutdown_logout_client.py
├─ mac_address_client.py
├─ requirements.txt
├─ registry_client.py
├─ client.py
├─ webcam_client.py
├─ live_screen_client.py
└─ directory_tree_client.py
```

Hình 2: Tổ chức thư mục ở client

Tập tin **requirements.txt** chứa các yêu cầu cần thiết (bên cạnh Python 3.9 trở lên) để chạy chương trình.

Trừ tập tin **client.py**, các tập tin mã nguồn Python còn lại chứa các hàm xử lý các yêu cầu được gửi đến từ server.

Tập tin **client** là tập tin chính của client, ta cần chạy tập tin này để khởi động client. **Lưu ý** khi chạy tập tin này ta cần thay đổi giá trị 127.0.0.1 ở dòng 60 thành một giá trị thích hợp.

Tất cả các client phải được kết nối đến server TRƯỚC khi server bắt đầu nhận và xử lý các lệnh từ email.

3 Cú pháp email

Các email điều khiển được gửi về địa chỉ **notabotbytheway@outlook.com**.

Các lệnh phải được viết ở **tiêu đề** của email.

Giá trị key là **1234**.

Cú pháp của các lệnh điều khiển được liệt kê bên dưới:

- **AUTH** <key> <IP>: "đăng ký" địa chỉ email để điều khiển <IP>. Tại một thời điểm, mỗi địa chỉ email đăng ký **duy nhất** với một địa chỉ IP và ngược lại.
- **LIST** <key>: liệt kê danh sách các IP đang kết nối.
- **DISC**: ngắt kết nối địa chỉ email này với IP hiện tại.
- **LIST_PROC**: liệt kê danh sách các tiến trình (process).
- **LIST_APP**: liệt kê danh sách các ứng dụng (application).
- **KILL** <ID/PID>: tắt (kill) tiến trình có mã là <ID/PID>.
- **SCREENSHOT** <time>: chụp màn hình liên tục trong thời gian <time> giây, mặc định là 0.5 giây.
- **WEB/REC** <time>: quay lại webcam của máy trong thời gian <time> giây, mặc định là 5 giây.
- **KEYLOG** <time>: bắt các phím nhấn trong thời gian <time> giây, mặc định là 10 giây.
- **SHUTDOWN**: tắt máy tính.
- **LOGOUT**: đăng xuất.
- **RESTART**: khởi động lại máy tính.
- **REGISTRY LIST** <path>: danh sách các registry trong path.
- **REGISTRY UPDATE** <registry path> <value> <data type>: cập nhật giá trị của <registry path> thành <value> với kiểu <data type>. <value> **không** được có khoảng trắng. <data type> được lấy từ [link](#).

- **DIR LIST** <path>: liệt kê các thư mục/file trong đường dẫn <path>.
- **DIR COPY** <source file> <destination path>: copy file có tên <source file> đến đường dẫn <destination path>.

4 Gửi và nhận email

4.1 Nhận mail

- Giao thức sử dụng: POP3.
- Nhà cung cấp mail: Microsoft Outlook.
- Sử dụng đa luồng để tối ưu hóa tác vụ.

Các hàm sử dụng cho quá trình nhận mail (POP3_service.py):

```
def get_mails():
```

Chức năng: Lấy toàn bộ mail đang có trong mail box, tách lấy *email người gửi*, *tiêu đề* và *nội dung* (nếu có), đưa vào hàng đợi để luồng khác xử lý, sau đó thì xóa mail để đảm bảo không trùng lặp với thư cũ.

Quá trình thực hiện:

- Tạo kết nối POP3 tới server (có sử dụng SSL).
- Gửi username (mail) và password để xác thực.
- Lấy số lượng mail hiện có qua lệnh LIST.
- Lấy thông tin từng thư qua lệnh RETR.
- Tách lấy thông tin.
- Xóa thư với lệnh DELE.
- Gửi lệnh QUIT ngắt kết nối.

```
def loop():
```

Chức năng: Do tính chất của POP3 không cập nhật thư mới, cần phải reload lại sau một khoảng thời gian, nên hàm này sẽ thực hiện việc lấy thư liên tục (gọi tới hàm `get_mails()`) khi chương trình khởi chạy.

4.2 Gửi mail

- Giao thức sử dụng: SMTP.
- Nhà cung cấp mail: Microsoft Outlook.
- Sử dụng đa luồng để tối ưu hóa tác vụ.

Các hàm sử dụng cho quá trình gửi mail (SMTP_service.py):

```
def send(to_: str, subject_: str, content_, file_name):
```

Chức năng: Gửi thư đến địa chỉ `to_`, với tiêu đề `subject_`, nội dung `content_` và tên file `file_name` nếu nội dung cần gửi là file (file ảnh, video...)

Tham số:

- `to_`: kiểu `str`, là địa chỉ mail người nhận (ví dụ `abc@gmail.com`).
- `subject_`: kiểu `str`, là tiêu đề của mail.
- `content_`: kiểu `str` hoặc `bytes`. Nếu là `str`, thực hiện gửi như văn bản bình thường. Nếu là `bytes`, gửi như một file với tên file là `file_name`.
- `file_name`: kiểu `str`, tên file (mặc định là `"x.txt"`).

Quá trình thực hiện:

- Tạo kết nối SMTP tới server (có sử dụng TLS) .
- Gửi username (mail) và password để xác thực.
- Soạn thư với định dạng **MIME multipart**.
- Nếu thư kiểu text, attach MIME text.
- Nếu thư kiểu bytes, attach MIME application với tên file.
- Gửi payload đến server và đóng kết nối.

```
def safe_send(to_:str, subject_:str, content_, file_name):
```

Chức năng: Là wrapper cho hàm `send`, thực hiện xử lý ngoại lệ: thực hiện gửi thư lại cho đến khi thành công (do trong một số trường hợp, gửi mail quá nhanh server sẽ từ chối nên phải thực hiện lại).

Tham số: tương tự `send`.

```
def send_threading(to_:str, subject_:str, content_, file_name = "x.txt"):
```

Chức năng: tạo threading để gửi thư, các tham số sẽ truyền trực tiếp vào cho hàm `safe_send`. (Đây là hàm sẽ được gọi để gửi thư khi được import)

5 Các hàm xử lý yêu cầu

5.1 Phân tích câu lệnh

Hàm phân tích câu lệnh được định nghĩa là:

```
def command_parser(message, sender_email_address):
```

Chức năng: phân tích câu lệnh được gửi đến, gọi hàm tương ứng với yêu cầu của câu lệnh.

Tham số:

- `message`: câu lệnh được gửi đến
- `sender_email_address`: địa chỉ email của người gửi.

Không có giá trị trả về.

Hàm `get_corresponding_ip`

```
def get_corresponding_ip(sender_email_address):
```

Chức năng: lấy ra địa chỉ IP tương ứng với `sender_email_address`.

Tham số:

- `sender_email_address`: địa chỉ email của người gửi.

Trả về địa chỉ IP tương ứng, trả về None nếu không tồn tại.

5.2 Module `mail_handler`

Các hàm được định nghĩa trong module `handle.py` được sử dụng để xử lý các yêu cầu từ người dùng.

```
def authorize(email, ip):
```

Chức năng: authorize địa chỉ email với IP tương ứng.

Tham số:

- `email`: địa chỉ email.
- `ip`: kiểu `tuple(str, int)` chứa địa chỉ ip và port của client gửi tới.

Trả về True nếu thành công, False nếu không thành công.

```
def list_ip():
```

Chức năng: trả về danh sách các địa chỉ IP đang được kết nối.

```
def disconnect(email):
```

Chức năng: disconnect địa chỉ email hiện tại với IP tương ứng. Tham số:

- `email`: địa chỉ IP tương ứng.

Không có giá trị trả về.

```
def find_corresponding_email(ip_address):
```

Chức năng: tìm địa chỉ email tương ứng với IP được truyền vào. Tham số:

- `ip_address`: kiểu `tuple(str, int)` chứa địa chỉ ip và port của client gửi tới.

Trả về: địa chỉ email tương ứng đang điều khiển địa chỉ IP này.

```
def delete_ip_from_list(ip_address):
```

Chức năng: xóa địa chỉ IP ra khỏi danh sách IP kết nối. Tham số:

- `ip_address`: kiểu tuple(str, int) chứa địa chỉ ip và port của client gửi tới.

Không có giá trị trả về.

```
def remove_this_connection(conn, ip_address):
```

Chức năng: loại bỏ kết nối tương ứng với IP. Tham số:

- `conn`: socket tương ứng.
- `ip_address`: kiểu tuple(str, int) chứa địa chỉ ip và port của client gửi tới.

Không có giá trị trả về.

```
def __init__():
```

Chức năng: khởi tạo những giá trị liên quan.

5.3 Process/Application

Server sẽ gửi lệnh **APP_PRO** đến client, sau đó sẽ gửi số tương ứng với lệnh.

- Gửi số 1 nếu ta cần đưa ra danh sách process/application. Sau đó gửi message là **APPLICATION** hoặc **PROCESS** tương ứng với yêu cầu của lệnh đến client. Client nhận thông điệp rồi gửi lại danh sách application/process dưới dạng một data frame.
- Gửi số 0 nếu ta cần kill một process/application. Sau đó ta sẽ gửi ID/PID của đối tượng cần kill. Client nhận thông điệp rồi gửi thông tin về quá trình kill lại cho server.

Các hàm cho quá trình này như sau:

- Ở server (mail_handler.py, app_process_server.py):

```
def list_process(ip_address):
```

Chức năng: gửi yêu cầu đến client, nhận kết quả rồi gửi mail lại cho người dùng.
Tham số:

- `ip_address`: kiểu tuple(str, int) chứa địa chỉ ip và port của client gửi tới.

Trả về: Không có giá trị trả về.

```
def _list(conn:socket.socket, s):
```

Chức năng: gửi yêu cầu danh sách tiến trình/ứng dụng đến client, nhận kết quả từ client.

Tham số:

- `conn`: socket kết nối server với client.
- `s`: chuỗi, có giá trị là **PROCESS** hoặc **APPLICATION**, thể hiện yêu cầu gửi danh sách tiến trình hoặc ứng dụng.

Trả về: Danh sách tiến trình/ứng dụng ở dạng chuỗi.

```
def send_kill(conn:socket.socket, process_id):
```

Chức năng: gửi yêu cầu tắt một tiến trình.

Tham số:

- `conn`: socket kết nối server với client.
- `process_id`: id của tiến trình cần tắt.

Trả về: kết quả kill thành công/không thành công.

Thư viện sử dụng: pickle, struct, pandas.

- Ở client (`app_process_client.py`):

```
def app_process(client):
```

Chức năng: nhận yêu cầu từ server, xử lý yêu cầu rồi gửi lại kết quả tương ứng.

Tham số:

- `client`: socket kết nối server với client.

Trả về: Không có giá trị trả về.

Thư viện sử dụng: pickle, psutil, struct, os, subprocess.

Bên cạnh đó, server còn có các hàm (được cung cấp sẵn) phụ trách việc nhận dữ liệu:

```
def recvall(sock, size):  
def receive(client):
```

Ngoài ra, client còn có các hàm (được cung cấp sẵn) phục vụ việc gửi dữ liệu cho server và xử lý các yêu cầu về application/process trên máy.

```
def send_data(client, data):  
def list_apps():  
def list_processes():  
def kill(pid):
```

5.4 Chụp màn hình

Server sẽ gửi lệnh **LIVESCREEN** đến client, sau đó sẽ gửi lệnh tương ứng với thời gian cần chụp màn hình, thời gian mặc định là 0.5 giây. Thời gian chụp không nên quá nhiều, vì giới hạn dung lượng tệp đính kèm trong email. Sau đó, server sẽ nhận ảnh gửi từ client, tạo thành một video.

Các hàm cho quá trình này như sau:

- Ở server (mail_handler.py):

```
def capture_screen(ip_address, time=0.5):
```

Chức năng: gửi yêu cầu chụp màn hình đến cho client, nhận hình ảnh từ client gửi lại, tạo video, gửi mail trả lời cho người dùng.

Tham số:

- `ip_address`: kiểu tuple(str, int) chứa địa chỉ ip và port của client gửi tới.
- `time`: thời gian chụp màn hình (giây).

Không có giá trị trả về.

- Ở client:

```
def capture_screen(client):
```

Chức năng: nhận yêu cầu chụp màn hình từ server, chụp màn hình liên tục rồi gửi từng ảnh lại cho server.

Tham số:

- `client`: socket kết nối server với client.

Không có giá trị trả về.

Thư viện sử dụng: ImageGrab, io, time.

Ngoài ra, ở server còn có hàm sau dùng để tạo video từ những ảnh chụp màn hình nhận được từ client, sử dụng thư viện os, cv2.

```
def create_video(image_folder: str):
```

5.5 Webcam

Server sẽ gửi lệnh **WEBCAM** đến client, sau đó sẽ gửi lệnh tương ứng với thời gian cần chụp màn hình, thời gian mặc định là 5 giây. Client nhận thông điệp từ server, quay màn hình webcam trong khoảng thời gian đó, rồi gửi lại video cho server.

Các hàm cho quá trình này như sau:

- Ở server (mail_handler.py):

```
def capture_webcam(ip_address, time=5):
```

Chức năng: gửi yêu cầu ghi lại webcam đến cho client, nhận video từ client gửi về, gửi mail trả lời cho người dùng.

Tham số:

- `ip_address`: kiểu `tuple(str, int)` chứa địa chỉ ip và port của client gửi tới.
- `time`: thời gian ghi hình webcam (giây).

Không có giá trị trả về.

- Ở client (`webcam_client.py`):

```
def run(conn: socket.socket):
```

Chức năng: nhận yêu cầu từ server, quay màn hình webcam, gửi lại file video cho server.

Tham số:

- `conn`: socket kết nối server với client.

Không có giá trị trả về.

Thư viện sử dụng: `tempfile`, `cv2`, `time`.

Ngoài ra, ở client còn có hàm sau dùng để gửi file về cho server:

```
def send_file(conn: socket.socket):
```

5.6 Keylogger

Server sẽ gửi lệnh **KEYLOG** cho client. Do thiết kế của các hàm có sẵn cho việc keylog ở client, server sẽ gửi 2 lệnh **HOOK** đến client để lấy các hành động giữa 2 lần. Sau đó, một lệnh **PRINT** được gửi đến client để client gửi dữ liệu về các phím nhấn đã bắt được cho server. Server nhận kết quả từ client.

Các hàm cho quá trình này như sau:

- Ở server (`mail_handler.py`):

```
def keylog(ip_address, time=10):
```

Chức năng: gửi yêu cầu keylog đến client, nhận kết quả từ client gửi về, gửi mail trả lời cho người dùng.

Tham số:

- `ip_address`: kiểu `tuple(str, int)` chứa địa chỉ ip và port của client gửi tới.
- `time`: thời gian keylog (giây).

Không có giá trị trả về.

Sử dụng hàm `sleep` của thư viện `time`.

- Ở client (`keylogger_client.py`):

```
def keylog(client):
```

Chức năng: nhận lệnh từ server, bắt các phím nhấn rồi gửi kết quả lại cho server, ngoài ra còn có chức năng phụ để có thể khóa phím người dùng (Không được đề cập trong chương trình).

Tham số:

- `client`: socket kết nối server với client.

Không có giá trị trả về.

Thư viện sử dụng: `threading`, `keyboard`, `pynput.keyboard`.

Ngoài ra, ở client còn có các hàm để hỗ trợ cho hàm `keylog` trong việc bắt phím nhấn cũng như gửi kết quả lại cho server.

```
def keylogger(key):
```

Chức năng: Dựa vào `global flag` (nhận giá trị 0, 1, 2, 4), thực hiện xử lý `key` lấy được từ `Listener` trong hàm `listen()` như sau:

- + Thay thế `space` thành ký tự khoảng trắng
- + Thay thế phím `'` thành chuỗi phù hợp
- + Lọc bỏ toàn bộ ký tự `'` dư thừa
- + Nối vào câu

Tham số:

- `key`: Phím được bắt từ quá trình `Listener`

Không có giá trị trả về.

```
def _print(client):
```

Chức năng: Gửi câu đang được ghi về server

Tham số:

- `client`: socket kết nối server với client..

Không có giá trị trả về.

```
def listen():
```

Chức năng: Tạo thread mới để ghi bàn phím (sử dụng hàm `keylogger` để xử lý)

Không có giá trị trả về.

5.7 Registry

Sẽ có 2 loại lệnh được gửi cho client:

- Liệt kê subkey: SERVER sẽ gửi lệnh **REGISTRY** cùng với **LIST** và đường dẫn (ví dụ như là `HKEY_CURRENT_USER`, `HKEY_CURRENT_USER/System`), client sẽ trả về danh sách các subkey tương ứng.

- Cập nhật key: SERVER sẽ gửi lệnh **REGISTRY** cùng với **UPDATE** và đường dẫn tới key, giá trị mới của key, và kiểu dữ liệu (kiểu dữ liệu thường thuộc 1 trong 4 loại: REG_BINARY, REG_DWORD, REG_QWORD, REG_SZ).

Các hàm cho quá trình này như sau:

- Ở server (mail_handler.py):

```
def registry_list(ip_address, full_path):
```

Chức năng: gửi các câu lệnh để liệt kê subkey với đường dẫn là full_path đến client, sau đó nhận về danh sách subkey, gửi mail trả lời cho người dùng.

Tham số:

- ip_address: kiểu tuple(str, int) chứa địa chỉ ip và port của client gửi tới.
- full_path: kiểu str, là đường dẫn cần liệt kê subkey.

Không có giá trị trả về.

```
def registry_update(ip_address, absolute_path, value, data_type):
```

Chức năng: gửi câu lệnh để cập nhật giá trị entry tới client, nhận về kết quả, gửi mail trả lời người dùng.

Tham số:

- ip_address: kiểu tuple(str, int) chứa địa chỉ ip và port của client gửi tới.
- absolute_path: đường dẫn tới entry cần cập nhật giá trị.
- value: giá trị mới cho entry, giá trị này KHÔNG được có khoảng trắng.
- data_type: kiểu dữ liệu của entry cần cập nhật.

Không có giá trị trả về.

- Ở client (registry_client.py):

```
def registry_handle(conn):
```

Chức năng: xử lý các yêu cầu gửi đến rồi trả kết quả lại cho server.

Tham số:

- conn: socket kết nối server với client.

Trả về:

```
def identify_hkey(value_list):
```

Chức năng: Xác định hive từ đường dẫn đã được tách

Tham số:

- ip_address: kiểu tuple(str, int) chứa địa chỉ ip và port của client gửi tới.

- `full_path`: kiểu `str`, là đường dẫn cần liệt kê subkey.

Trả về: giá trị hive tương ứng.

```
def get_value_of_key(key):
```

Chức năng: lấy giá trị của `key`.

Tham số:

- `key`: key cần lấy giá trị.

Trả về: giá trị tại `key`.

```
def get_sub_keys(key):
```

Chức năng: lấy ra các subkeys của `key`.

Tham số:

- `key`: key cần lấy các subkeys.

Trả về: subkeys của `key`.

```
def list_all_registry_entries(registry_path, reg_dict):
```

Chức năng: liệt kê danh sách các entries trong 1 đường dẫn registry.

Tham số:

- `registry_path`: đường dẫn registry cần liệt kê.
- `reg_dict`: dictionary dùng để lưu kết quả.

Không có giá trị trả về.

Ngoài ra, client có các hàm được cung cấp sẵn để giúp cho quá trình xử lý yêu cầu trên registry.

```
def parse_data(full_path):  
def dec_value(c):  
def str_to_bin(s):  
def str_to_dec(s):  
def set_value(full_path, value, value_type):
```

5.8 Directory

Sẽ có 2 loại lệnh được gửi cho client:

- Liệt kê files/directories: SERVER sẽ gửi lệnh **DIR** cùng với **LIST** và đường dẫn (ví dụ như là `C:/Users/admin`), client sẽ trả về danh sách các files/directories trong đường dẫn tương ứng.
- Cập nhật key: SERVER sẽ gửi lệnh **DIR** cùng với **COPY**, đường dẫn tới file cần copy cũng như thư mục mà file sẽ được copy vào.

Lưu ý: các đường dẫn KHÔNG được có khoảng trắng.
Các hàm cho quá trình này như sau :

- Ở server (mail_handler.py):

```
def dir_list(ip_address, path_to_folder):
```

Chức năng: gửi yêu cầu liệt kê danh sách files/directories đến client, nhận kết quả, gửi mail trả lời người dùng.

Tham số:

- ip_address: kiểu tuple(str, int) chứa địa chỉ ip và port của client gửi tới.
- path_to_folder: đường dẫn cần liệt kê.

Không có giá trị trả về.

```
def dir_copy(ip_address, src_path, dst_path):
```

Chức năng: gửi yêu cầu copy file đến client, nhận kết quả, gửi mail trả lời người dùng.

Tham số:

- ip_address: kiểu tuple(str, int) chứa địa chỉ ip và port của client gửi tới.
- src_path: đường dẫn tới file cần copy.
- dst_path: thư mục mà file sẽ được copy vào.

Không có giá trị trả về.

- Ở client (directory_tree_client.py):

```
def directory_handle(conn):
```

Chức năng: xử lý các yêu cầu về liệt kê/copy từ server, gửi kết quả lại cho server.

Tham số:

- conn: socket kết nối server với client.

Không có giá trị trả về.

5.9 Shutdown/log out/restart

Server sẽ gửi lệnh **SHUTDOWN** hoặc **LOGOUT** hoặc **RESTART**, tùy theo yêu cầu, đến client. Client nhận lệnh rồi thực hiện. Khi client thực hiện nhóm lệnh này, kết nối của client tương ứng với server sẽ bị ngắt, đồng thời email đang điều khiển client này cũng bị disconnect, nếu muốn kết nối lại thì phải authorize lại.

Các hàm cho quá trình này như sau:

- Ở server:
mail_handler.py:

```
def shut_down(ip_address):  
def logout(ip_address):  
def restart(ip_address):
```

Chức năng: gọi hàm gửi yêu cầu đến client, gửi mail xác nhận cho người dùng.
Tham số:

- `ip_address`: kiểu `tuple(str, int)` chứa địa chỉ ip và port của client gửi tới.

Không có giá trị trả về.

`shutdown_logout_server.py`:

```
def shutdown(conn):  
def logout(conn):  
def restart(conn):
```

Chức năng: gửi yêu cầu tương ứng đến client.
Tham số:

- `conn`: socket kết nối server với client.

Không có giá trị trả về.

- Ở client (`shutdown_logout_client.py`):

```
def shutdown_logout(conn, msg):
```

Chức năng: xử lý yêu cầu shutdown/log out/restart từ server.
Tham số:

- `conn`: socket kết nối server với client.
- `msg`: thông điệp gửi từ server.

Không có giá trị trả về.

6 Demo

Video demo của chương trình được đăng tải tại [đây](#).

Lời cảm ơn

Trong quá trình thực hiện đồ án này, những kiến thức về Mạng máy tính được giảng dạy bởi thầy Đỗ Hoàng Cường đã giúp ích chúng em rất nhiều trong việc giải quyết các vấn đề. Ngoài ra, source code TelePC từ thầy đã giúp nhóm có thể dễ dàng hơn khi xử lý các yêu cầu của đồ án. Nhóm chúng em xin cảm ơn thầy vì những kiến thức cũng như những chia sẻ này ạ.

Bên cạnh đó, những kỹ năng lập trình socket được các giáo viên hướng dẫn thực hành là thầy Lê Hà Minh và thầy Nguyễn Thanh Quân truyền đạt đã góp phần không nhỏ trong việc hoàn thành đồ án này.

Ngoài ra, nhóm cũng xin chân thành cảm ơn những người bạn đã đóng góp ý kiến, góp phần hoàn thiện đồ án.

Thành phố Hồ Chí Minh, tháng 6 năm 2022