

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

---o0o---



BÁO CÁO
ĐỒ ÁN LẬP TRÌNH SOCKET:
ĐIỀU KHIỂN TỪ XA BẰNG EMAIL

Thành viên nhóm	MSSV
Đỗ Phan Tuấn Đạt	22127057
Phạm Thành Đạt	22127064
Lê Hồ Phi Hoàng	22127123
Trần Nguyễn Minh Hoàng	22127131

Lớp: 22CLC02

Môn học: Mạng máy tính

Học kỳ: 1

Năm học: 2023-2024

I. Mục lục

I.	Mục lục.....	2
II.	Môi trường làm việc, thư viện hỗ trợ.....	4
1.	Môi trường làm việc:	4
2.	Ngôn ngữ:	4
3.	Thư viện hỗ trợ:	4
III.	Danh sách file trong source code.....	4
IV.	Phân tích, giải thích source code	5
1.	main.py	5
2.	readMail.py	11
3.	sendMail.py	19
4.	keyLog.py	29
5.	appController.py	31
6.	processController.py.....	36
7.	powerController.py.....	39
V.	Hướng dẫn sử dụng	40
1.	Thiết lập ứng dụng điều khiển từ xa.....	40
a)	Thiết bị được điều khiển.....	40
b)	Thiết bị điều khiển.....	40
2.	Ghi bàn phím	40
3.	Chụp màn hình.....	41
4.	Lấy danh sách các app đang hoạt động	42
5.	Lấy danh sách các process đang hoạt động	43
6.	Khởi động app	44
7.	Tắt app	45
8.	Khởi động process	45
9.	Tắt process	46
10.	Đăng xuất khỏi Windows	47
11.	Tắt nguồn.....	47

12.	Tắt ứng dụng điều khiển từ xa.....	47
c)	Cách 1: Tắt trên thiết bị được điều khiển.....	47
d)	Cách 2: Tắt bằng email.....	47
VI.	Đóng góp.....	48
VII.	Tài liệu tham khảo	48

II. Môi trường làm việc, thư viện hỗ trợ

1. Môi trường làm việc:

Visual Studio Code

2. Ngôn ngữ:

Python

3. Thư viện hỗ trợ:

- **tkinter:** Tạo giao diện người dùng
- **os:** Cho chức năng hệ điều hành
- **signal:** Gửi tín hiệu đến các tiến trình
- **psutil:** Quản lý tiến trình
- **subprocess:** Chạy lệnh bên ngoài
- **tabulate:** Định dạng dữ liệu thành bảng
- **bs4:** Phân tích cú pháp HTML và XML
- **imaplib:** Truy cập và thao tác với email trên server IMAP
- **email:** Làm việc với email
- **email.header:** Giải mã email có thể chứa ký tự không phải ASCII
- **threading:** Tạo và quản lý luồng
- **time:** Làm việc với thời gian
- **re:** Tìm kiếm và thao tác chuỗi dựa theo mẫu
- **smtplib:** Gửi email
- **PIL:** Chụp màn hình
- **windll:** import từ module **ctypes** để sử dụng các hàm của thư viện động Windows.

III. Danh sách file trong source code

- **main.py:** Chứa các hàm để tạo giao diện cho chương trình và chạy chương trình
- **readMail.py:** Chứa các hàm để đọc email và thực hiện lệnh theo từ khóa trong nội dung mail
- **sendMail.py:** Chứa hàm để gửi email reply tương ứng với lệnh đã thực hiện
- **keyLog.py:** Chứa class keyLog dùng cho chức năng ghi bàn phím
- **appController.py:** Chứa class AppController cùng các hàm dùng trong các hoạt động liên quan đến app
- **processController.py:** Chứa class ProcessController cùng các hàm dùng trong các hoạt động liên quan đến process

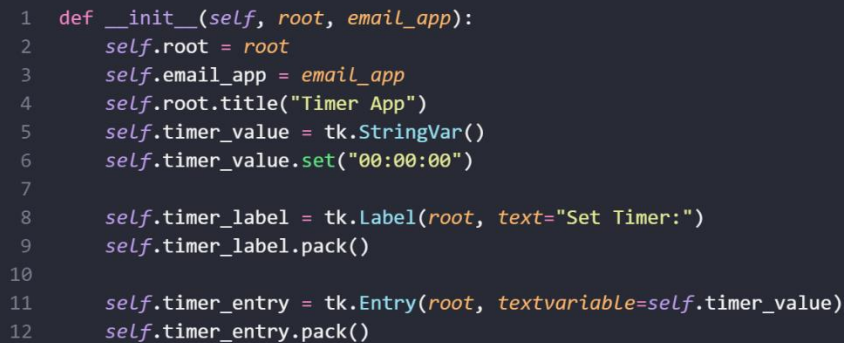
- `powerController.py`: Chứa các hàm để đăng xuất khỏi Window và tắt thiết bị

IV. Phân tích, giải thích source code

1. `main.py`

Class **TimerApp** chứa các hàm để tính thời gian chương trình hoạt động

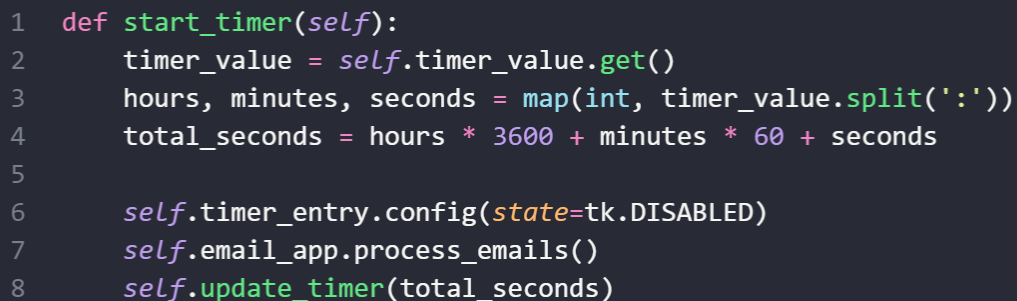
- `__init__`:



```
1 def __init__(self, root, email_app):
2     self.root = root
3     self.email_app = email_app
4     self.root.title("Timer App")
5     self.timer_value = tk.StringVar()
6     self.timer_value.set("00:00:00")
7
8     self.timer_label = tk.Label(root, text="Set Timer:")
9     self.timer_label.pack()
10
11     self.timer_entry = tk.Entry(root, textvariable=self.timer_value)
12     self.timer_entry.pack()
```

- + **`self.root = root`**: Lưu trữ tham chiếu đến cửa sổ chính của ứng dụng Tkinter.
- + **`self.email_app = email_app`**: Lưu trữ tham chiếu đến đối tượng **EmailProcessingApp** để giao tiếp giữa các phần của ứng dụng.
- + **`self.root.title("Timer App")`**: Thiết lập tiêu đề của cửa sổ ứng dụng là "Timer App".
- + **`self.timer_value = tk.StringVar()`**: Tạo một đối tượng **StringVar** trong Tkinter để lưu trữ và cập nhật giá trị thời gian.
- + **`self.timer_value.set("00:00:00")`**: Khởi tạo giá trị mặc định cho thời gian là "00:00:00".
- + **`self.timer_label = tk.Label(root, text="Set Timer:")`**: Tạo nhãn (Label) để hiển thị thông báo "Set Timer:" trên giao diện.
- + **`self.timer_label.pack()`**: Đặt nhãn vào cửa sổ giao diện.
- + **`self.timer_entry = tk.Entry(root, textvariable=self.timer_value)`**: Tạo một ô nhập liệu (Entry) trong giao diện, liên kết với đối tượng **StringVar** để hiển thị và cho phép người dùng nhập giá trị thời gian.
- + **`self.timer_entry.pack()`**: Đặt ô nhập liệu vào cửa sổ giao diện.

- *start_timer:*

A code block with a dark background and light-colored text. At the top left of the code block are three colored circles: red, yellow, and green. The code is a Python function definition for `start_timer` within a class (indicated by `self`). It takes the current timer value, splits it into hours, minutes, and seconds, calculates the total seconds, disables the input field, processes emails, and updates the timer.

```
1 def start_timer(self):  
2     timer_value = self.timer_value.get()  
3     hours, minutes, seconds = map(int, timer_value.split(':'))  
4     total_seconds = hours * 3600 + minutes * 60 + seconds  
5  
6     self.timer_entry.config(state=tk.DISABLED)  
7     self.email_app.process_emails()  
8     self.update_timer(total_seconds)
```

- + **timer_value = self.timer_value.get()**: Lấy giá trị thời gian nhập vào từ thành phần giao diện người dùng.
- + **hours, minutes, seconds = map(int, timer_value.split(':'))**: Tách giá trị thời gian thành các giờ, phút và giây.
- + **total_seconds = hours * 3600 + minutes * 60 + seconds**: Chuyển đổi thời gian thành tổng số giây.
- + **self.timer_entry.config(state=tk.DISABLED)**: Vô hiệu hóa ô nhập liệu thời gian để người dùng không thể thay đổi thời gian trong khi đếm ngược đang diễn ra.
- + **self.email_app.process_emails()**: Gọi phương thức **process_emails** của đối tượng **email_app**, cho phép xử lý email trong khi bắt đầu đếm ngược.
- + **self.update_timer(total_seconds)**: Bắt đầu đếm ngược với tổng số giây được truyền vào.

- *update_timer:*

```

1
2 def update_timer(self, remaining_seconds):
3     if remaining_seconds >= 0:
4         hours = remaining_seconds // 3600
5         minutes = (remaining_seconds % 3600) // 60
6         seconds = remaining_seconds % 60
7         timer_value = f"{hours:02d}:{minutes:02d}:{seconds:02d}"
8         self.timer_value.set(timer_value)
9
10        self.root.after(1000, self.update_timer, remaining_seconds - 1)
11    else:
12        self.timer_entry.config(state=tk.NORMAL)

```

- + **if remaining_seconds >= 0:** Kiểm tra nếu thời gian còn lại lớn hơn hoặc bằng 0.
- + **hours = remaining_seconds // 3600:** Tính toán số giờ còn lại từ tổng số giây.
- + **minutes = (remaining_seconds % 3600) // 60:** Tính toán số phút còn lại từ số giây còn lại sau khi loại bỏ giờ.
- + **seconds = remaining_seconds % 60:** Tính toán số giây còn lại.
- + **timer_value = f"{hours:02d}:{minutes:02d}:{seconds:02d}":** Tạo chuỗi thời gian mới với định dạng HH:MM:SS.
- + **self.timer_value.set(timer_value):** Cập nhật giá trị thời gian trong đối tượng **StringVar** để hiển thị trên giao diện.
- + **self.root.after(1000, self.update_timer, remaining_seconds - 1):** Sử dụng **after** để lên lịch gọi lại chính nó sau 1000ms (1 giây) với thời gian còn lại giảm đi 1 giây, tạo hiệu ứng đếm ngược. Nếu thời gian còn lại là 0, đếm ngược kết thúc và ô nhập liệu thời gian được kích hoạt trở lại để người dùng có thể thay đổi thời gian.

Class **EmailProcessingApp** chứa các hàm tạo ra giao diện người dùng của chương trình cũng như chạy chương trình

- **__init__:**


```

1  def __init__(self, root):
2      self.root = root
3      self.root.title("Email Processing App")
4      self.is_running = False
5      self.is_first_run = True # Flag to indicate the first run
6      self.progress_value = 0
7
8      # Create GUI elements
9      self.start_stop_button = tk.Button(root, text="Start", command=self.start_timer)
10     self.start_stop_button.pack(pady=20)
11
12     self.progress_label = tk.Label(root, text="Command Progress:")
13     self.progress_label.pack()
14
15     self.progress_bar = ttk.Progressbar(root, length=200, mode="determinate")
16     self.progress_bar.pack()
17
18     self.command_label = tk.Label(root, text="Command: ")
19     self.command_label.pack()
20
21     # Create TimerApp instance
22     self.timer_app = TimerApp(root, self)

```

- + **self.root = root**: Lưu trữ tham chiếu đến cửa sổ chính của ứng dụng Tkinter.
- + **self.root.title("Email Processing App")**: Thiết lập tiêu đề của cửa sổ là "Email Processing App".
- + **self.is_running = False**: Biến đánh dấu xem quá trình xử lý email đang chạy hay không. Ban đầu được đặt là **False**.
- + **self.is_first_run = True**: Cờ đánh dấu cho biết đây có phải lần chạy đầu tiên không. Trạng thái ban đầu là **True**.
- + **self.start_stop_button**: Button để bắt đầu hoặc dừng xử lý email, khi nhấn sẽ gọi đến phương thức **start_timer**.
- + **self.progress_label**: Nhãn để hiển thị tiến độ của lệnh (command progress).
- + **self.progress_bar**: Thanh tiến trình để thể hiện tiến độ của công việc.
- + **self.command_label**: Nhãn để hiển thị thông tin về lệnh đang được thực thi.
- + **self.timer_app = TimerApp(root, self)**: Tạo một đối tượng **TimerApp** để quản lý tính năng đếm ngược thời gian.

- *start_timer*:



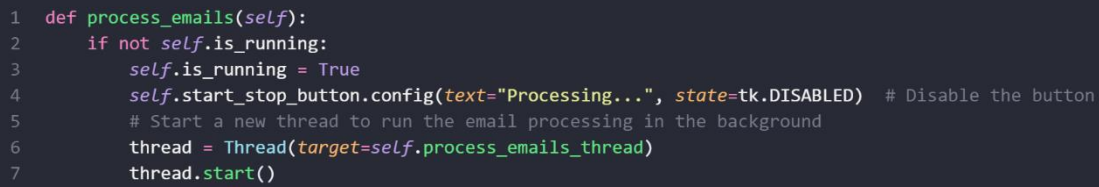
```

1  def start_timer(self):
2      self.timer_app.start_timer()

```

- + **self.timer_app.start_timer()**: Gọi phương thức **start_timer** từ đối tượng **TimerApp** để bắt đầu đếm ngược.

- *process_emails:*



```

1  def process_emails(self):
2      if not self.is_running:
3          self.is_running = True
4          self.start_stop_button.config(text="Processing...", state=tk.DISABLED) # Disable the button
5          # Start a new thread to run the email processing in the background
6          thread = Thread(target=self.process_emails_thread)
7          thread.start()

```

- + **def process_emails(self)**:: Phương thức để xử lý các email.
- + **if not self.is_running**:: Kiểm tra xem quá trình xử lý email có đang chạy hay không.
- + **self.is_running = True**: Đặt biến **is_running** thành **True** để bắt đầu quá trình xử lý.
- + **self.start_stop_button.config(text="Processing...", state=tk.DISABLED)**: Cấu hình button để hiển thị "Processing..." và vô hiệu hóa nó để tránh khả năng nhấn nút khi quá trình đang chạy.
- + **thread = Thread(target=self.process_emails_thread)**: Tạo một luồng mới để thực thi phương thức **process_emails_thread**.
- + **thread.start()**: Khởi động luồng để bắt đầu xử lý email trong nền.

- *process_emails_thread:*

```

1  def process_emails_thread(self):
2      # Start the timer
3      self.start_timer()
4      while self.timer_app.timer_value.get() != "00:00:00":
5          # Call the read_email() function here
6          command = readMail.read_email()
7          if command == "Invalid":
8              self.update_progress(0, "Invalid")
9              continue
10         # Run the email processing logic
11         for i in range(1, 101):
12             self.update_progress(i, "Executing " + command)
13             time.sleep(random.uniform(0.01, 0.1)) # Random sleep time between 0.01 and 0.1 seconds
14
15         self.update_progress(0, "Command Complete")
16         if command == "Die":
17             self.stop_processing()
18
19     self.stop_processing()

```

- + **self.start_timer()**: Bắt đầu đếm ngược thời gian từ **TimerApp**.
- + **command = readMail.read_email()**: Đọc email và lấy lệnh từ email.
- + **Xử lý lệnh và cập nhật tiến độ**: Hiển thị tiến độ xử lý lệnh trên thanh tiến trình.
- + **time.sleep(random.uniform(0.01, 0.1))**: Tạm dừng một khoảng thời gian ngẫu nhiên để mô phỏng việc xử lý lệnh.
- + **self.stop_processing()**: Kết thúc quá trình xử lý email khi thời gian đếm ngược kết thúc hoặc khi lệnh là "Die".

- **stop_processing:**

```


1  def stop_processing(self):
2      self.is_running = False
3      self.progress_value = 0
4      self.progress_bar["value"] = 0
5      self.command_label.config(text="Command: ")
6      self.start_stop_button.config(text="Start", state=tk.NORMAL) # Enable the button
7      self.timer_app.timer_value.set("00:00:00")
8

```

- + **self.is_running = False**: Đặt biến **is_running** thành **False** để kết thúc quá trình xử lý email.
- + **self.progress_value = 0**: Đặt giá trị tiến độ về 0.
- + **self.progress_bar["value"] = 0**: Thiết lập giá trị thanh tiến trình về 0.

- + **self.command_label.config(text="Command: ")**: Xóa thông tin lệnh đang thực thi.
- + **self.start_stop_button.config(text="Start", state=tk.NORMAL)**: Thiết lập lại nút bắt đầu để có thể nhấn và bắt đầu lại quá trình xử lý email.
- + **self.timer_app.timer_value.set("00:00:00")**: Đặt giá trị thời gian trong **TimerApp** về "00:00:00".

- *update_progress:*



```

1  def update_progress(self, value, command):
2      self.progress_value = value
3      self.progress_bar["value"] = self.progress_value
4      self.command_label.config(text="Command: " + command)

```

- + **self.progress_value = value**: Cập nhật giá trị tiến độ.
- + **self.progress_bar["value"] = self.progress_value**: Cập nhật giá trị thanh tiến trình trên giao diện.
- + **self.command_label.config(text="Command: " + command)**: Hiển thị thông tin lệnh đang thực thi lên giao diện người dùng.

2. readMail.py

- *extract_email_information:*

```

1 def extract_email_information(raw_email):
2     # Parse the email message
3     email_message = email.message_from_bytes(raw_email)
4
5     # Extract information from the email
6     subject, encoding = decode_header(email_message["Subject"])[0]
7     if isinstance(subject, bytes):
8         subject = subject.decode(encoding)
9     from_, encoding = decode_header(email_message["From"])[0]
10    if isinstance(from_, bytes):
11        from_ = from_.decode(encoding)
12
13    sender_email = email_message["From"]
14    match = re.search(r'<([>]+)>', sender_email)
15    if match:
16        sender_email = match.group(1)
17
18    # Print email details
19    print("Subject:", subject)
20    print("From:", from_)
21
22    # If the email message is multipart
23    if email_message.is_multipart():
24        for part in email_message.walk():
25            content_type = part.get_content_type()
26            content_disposition = str(part.get("Content-Disposition"))
27            try:
28                body = part.get_payload(decode=True).decode()
29            except:
30                pass
31            if content_type == "text/plain" and "attachment" not in content_disposition:
32                print(body)
33    return sender_email, body

```

- + **email_message = email.message_from_bytes(raw_email)**: Chuyển đổi dữ liệu email từ dạng byte sang đối tượng email bằng cách sử dụng **email.message_from_bytes**. Điều này cho phép ta tương tác với thông tin trong email như tiêu đề, người gửi và nội dung.
- + **subject, encoding = decode_header(email_message["Subject"])[0]**: Trích xuất thông tin về tiêu đề của email. Hàm **decode_header** được sử dụng để giải mã tiêu đề vì có thể có mã hóa đặc biệt trong tiêu đề. Nếu tiêu đề là dạng bytes, nó sẽ được giải mã bằng cách sử dụng encoding tương ứng.
- + **from_, encoding = decode_header(email_message["From"])[0]**: Trích xuất thông tin về người gửi của email. Tương tự như tiêu đề, thông tin người gửi cũng có thể được mã hóa và cần giải mã nếu là dạng bytes.
- + **sender_email = email_message["From"]**: Lấy địa chỉ email của người gửi từ email.

- + **match = re.search(r'<([>]+)>', sender_email):** Sử dụng biểu thức chính quy để tìm kiếm địa chỉ email trong dấu ngoặc nhọn (<...>). Nếu tìm thấy, **sender_email** sẽ được cập nhật với địa chỉ email được tìm thấy.
- + **print("Subject:", subject), print("From:", from_):** Hiển thị thông tin về tiêu đề và người gửi email. Đây là dữ liệu được in ra để kiểm tra khi phân tích email.
- + **if email_message.is_multipart()::** Kiểm tra xem email có nhiều phần không. Nếu có, đoạn mã sẽ lặp qua từng phần để trích xuất nội dung.
- + **for part in email_message.walk()::** Lặp qua từng phần trong email.
- + **content_type = part.get_content_type():** Lấy loại nội dung của phần email hiện tại.
- + **content_disposition = str(part.get("Content-Disposition")):** Lấy thông tin về việc bố trí nội dung của phần email hiện tại.
- + **try: ... except: ...:** Thử giải mã nội dung phần email hiện tại. Nếu không thể giải mã, nó sẽ bỏ qua lỗi và không làm gì cả.
- + **if content_type == "text/plain" and "attachment" not in content_disposition: ...:** Kiểm tra xem nội dung của phần email hiện tại có phải là văn bản thuần túy không và không phải là file đính kèm. Nếu đúng, in ra nội dung văn bản này.
- + **return sender_email, body:** Trả về địa chỉ email của người gửi và nội dung của email (body) cho việc xử lý tiếp theo.

- **key_log(sender_email, body, start, end):**

```

1 def key_log(sender_email, body, start, end):
2     duration_start = body.find("Duration:", start, end)
3     if duration_start != -1:
4         duration_start += len("Duration:") # Move the start index to the character after "Duration:"
5         duration_end = body.find("\n", duration_start, end) # Find the end of the number, which is a newline character
6         if duration_end != -1:
7             duration_str = body[duration_start:duration_end] # Extract the substring containing the duration as a string
8             # Remove unwanted characters
9             duration_str = ''.join(filter(str.isdigit, duration_str))
10            duration = int(duration_str)
11            sendMail.send_keylog_email(sender_email, duration)

```

- + **duration_start = body.find("Duration:", start, end):** Tìm vị trí xuất hiện đầu tiên của chuỗi "Duration:" trong phần của văn bản được chỉ định.
- + **if duration_start != -1: ...:** Kiểm tra xem chuỗi "Duration:" có được tìm thấy không.

- + **duration_start += len("Duration:")**: Di chuyển chỉ số bắt đầu đến ký tự sau "Duration:".
- + **duration_end = body.find("\n", duration_start, end)**: Tìm vị trí kết thúc của số liệu, thường là ký tự xuống dòng "\n".
- + **if duration_end != -1: ...**: Kiểm tra xem kết thúc của số liệu có được tìm thấy không.
- + **duration_str = body[duration_start:duration_end]**: Trích xuất chuỗi chứa thời lượng như một chuỗi ký tự.
- + **duration_str = ''.join(filter(str.isdigit, duration_str))**: Loại bỏ các ký tự không phải là số từ chuỗi số liệu.
- + **duration = int(duration_str)**: Chuyển chuỗi số liệu thành một số nguyên.
- + **sendMail.send_keyLog_email(sender_email, duration)**: Gửi email với thông tin về thời lượng và người gửi email (**sender_email**).

- **get_name_app_or_process(body, stringstart)**:

```

1  def get_name_app_or_process(body, stringstart):
2      start_index = body.find(stringstart)
3      Name = ""
4      if start_index != -1:
5          name_start = start_index + len(stringstart)
6          name_end = body.find("\n", name_start)
7
8          if name_end != -1:
9              Name = body[name_start:name_end].strip()
10             Name = BeautifulSoup(Name, "html.parser").text      # Remove HTML tags
11             Name = Name.strip()
12     return Name

```

Thực hiện việc trích xuất tên của app hoặc quá trình từ văn bản dựa trên chuỗi bắt đầu được chỉ định (**stringstart**).

- + **start_index = body.find(stringstart)**: Tìm vị trí xuất hiện đầu tiên của chuỗi bắt đầu được chỉ định (**stringstart**) trong văn bản.
- + **Name = ""**: Khởi tạo biến **Name** rỗng.
- + **if start_index != -1: ...**: Kiểm tra xem chuỗi bắt đầu được tìm thấy không.

- + **name_start = start_index + len(stringstart):** Xác định vị trí bắt đầu của tên app hoặc quá trình.
- + **name_end = body.find("\n", name_start):** Tìm vị trí kết thúc của tên, thường là ký tự xuống dòng "\n".
- + **if name_end != -1: ...:** Kiểm tra xem kết thúc của tên có được tìm thấy không.
- + **Name = body[name_start:name_end].strip():** Trích xuất tên app hoặc quá trình và loại bỏ khoảng trắng xung quanh nó.
- + **Name = BeautifulSoup(Name, "html.parser").text:** Loại bỏ các thẻ HTML từ tên (nếu có).
- + **return Name:** Trả về tên đã được xử lý.

- Các hàm *end_process(sender_email, body)*, *start_process(sender_email, body)*, *start_app(sender_email, body)*, *end_app(sender_email, body)*:

```

1  def start_process(sender_email, body):
2      procName = get_name_app_or_process(body, "StartProcess Name =")
3      if procName != "":
4          sendMail.send_startProc_status(sender_email, procName)
5
6  def end_process(sender_email, body):
7      procName = get_name_app_or_process(body, "EndProcess Name =")
8      if procName != "":
9          sendMail.send_endProc_status(sender_email, procName)
10
11 def start_app(sender_email, body):
12     appName = get_name_app_or_process(body, "StartApp Name =")
13     if appName != "":
14         sendMail.send_startApp(sender_email, appName)
15
16 def end_app(sender_email, body):
17     appName = get_name_app_or_process(body, "EndApp Name =")
18     if appName != "":
19         sendMail.send_EndApp(sender_email, appName)
20

```

Các hàm này sử dụng hàm **get_name_app_or_process()** để trích xuất tên app hoặc process từ văn bản, sau đó gửi email với thông tin tương ứng nếu tên được tìm thấy.

- *process_email:*

```
1 def process_email(sender_email, body):
2     start = 0 # Define the start position in the string
3     end = len(body) # Define the end position in the string
4
5     if body.find("KeyLog", start, end) != -1:
6         key_log(sender_email, body, start, end)
7         return "KeyLog"
8
9     if body.find("ListApp", start, end) != -1:
10        sendMail.send_process_email(sender_email)
11        return "ListApp"
12
13    if body.find("Screenshot", start, end) != -1:
14        sendMail.send_screenshot_email(sender_email)
15        return "Screenshot"
16
17    if body.find("Shut Down", start, end) != -1:
18        power_controller = powerController.powerController()
19        power_controller.shutdown()
20
21    if body.find("Log out", start, end) != -1:
22        power_controller = powerController.powerController()
23        power_controller.logout()
24
25    if body.find("ListProcess", start, end) != -1:
26        sendMail.send_bgProcess_email(sender_email)
27        return "ListProcess"
28
29    if body.find("StartProcess", start, end) != -1:
30        start_process(sender_email, body)
31        return "StartProcess"
32
33    if body.find("EndProcess", start, end) != -1:
34        end_process(sender_email, body)
35        return "EndProcess"
36
37    if body.find("StartApp", start, end) != -1:
38        start_app(sender_email, body)
39        return "StartApp"
40
41    if body.find("EndApp", start, end) != -1:
42        end_app(sender_email, body)
43        return "EndApp"
44
45    if body.find("Die", start, end) != -1:
46        return "Die"
47
48    else:
49        return "Invalid"
```


- + **start = 0** và **end = len(body)**: Xác định vị trí bắt đầu và kết thúc trong chuỗi **body**.
- + **if body.find("KeyLog", start, end) != -1: ...**: Kiểm tra xem chuỗi "KeyLog" có xuất hiện trong văn bản không. Nếu có, gọi hàm **key_log()** để xử lý thông tin về Keylog và trả về chuỗi "KeyLog".
- + **if body.find("ListApp", start, end) != -1: ...**: Tương tự, kiểm tra chuỗi "ListApp" có xuất hiện trong văn bản không. Nếu có, gửi email liệt kê các app đang hoạt động và trả về chuỗi "ListApp".
- + **if body.find("Screenshot", start, end) != -1: ...**: Kiểm tra chuỗi "Screenshot" có xuất hiện trong văn bản không. Nếu có, gửi email chứa ảnh chụp màn hình và trả về chuỗi "Screenshot".
- + **if body.find("Shut Down", start, end) != -1: ...**: Kiểm tra chuỗi "Shut Down" có xuất hiện trong văn bản không. Nếu có, thực hiện thao tác tắt máy.
- + **if body.find("Log out", start, end) != -1: ...**: Kiểm tra chuỗi "Log out" có xuất hiện trong văn bản không. Nếu có, thực hiện thao tác đăng xuất.
- + **if body.find("ListProcess", start, end) != -1: ...**: Kiểm tra chuỗi "ListProcess" có xuất hiện trong văn bản không. Nếu có, gửi email liệt kê các process đang hoạt động và trả về chuỗi "ListProcess".
- + **if body.find("StartProcess", start, end) != -1: ...**: Kiểm tra chuỗi "StartProcess" có xuất hiện trong văn bản không. Nếu có, gọi hàm **start_process()** để xử lý thông tin khởi động một process và trả về chuỗi "StartProcess".
- + **if body.find("EndProcess", start, end) != -1: ...**: Kiểm tra chuỗi "EndProcess" có xuất hiện trong văn bản không. Nếu có, gọi hàm **end_process()** để xử lý thông tin kết thúc một process và trả về chuỗi "EndProcess".
- + **if body.find("StartApp", start, end) != -1: ...**: Kiểm tra chuỗi "StartApp" có xuất hiện trong văn bản không. Nếu có, gọi hàm **start_app()** để xử lý thông tin khởi động một app và trả về chuỗi "StartApp".
- + **if body.find("EndApp", start, end) != -1: ...**: Kiểm tra chuỗi "EndApp" có xuất hiện trong văn bản không. Nếu có, gọi hàm **end_app()** để xử lý thông tin kết thúc một app và trả về chuỗi "EndApp".
- + **if body.find("Die", start, end) != -1: ...**: Kiểm tra chuỗi "Die" có xuất hiện trong văn bản không. Nếu có, trả về chuỗi "Die".
- + **else: ...**: Trường hợp mặc định, Nếu không có từ khóa nào được tìm thấy, hàm trả về chuỗi "Invalid".

- ***read_email:***

Thực hiện việc đăng nhập vào tài khoản Gmail, tìm và lấy email chưa đọc, rồi xử lý thông tin từ email cuối cùng được tìm thấy.

```
1 def read_email():
2     # Gmail account credentials
3     username = "2d2h.computernetwork.clc.fitus@gmail.com"
4     password = "jvys wjpg mmmx belp"
5
6     # Create an IMAP4 class with SSL
7     imap = imaplib.IMAP4_SSL("imap.gmail.com")
8
9     # Authenticate
10    imap.login(username, password)
11
12    # Select the mailbox (inbox in this case)
13    imap.select("inbox")
14
15    # Search for all unread emails and retrieve the latest one, if none found then exit
16    status, email_ids = imap.search(None, 'UNSEEN')
17    if not email_ids[0]:
18        print("No unread emails found.")
19        return "Error"
20    latest_email_id = email_ids[0].split()[-1]
21
22    # Fetch the email message by ID
23    status, email_data = imap.fetch(latest_email_id, "(RFC822)")
24
25    # Extract email information
26    sender_email, body = extract_email_information(email_data[0][1])
27
28    # Process email
29    command = process_email(sender_email, body)
30
31    # Close the connection
32    imap.logout()
33    return command
```

- + **username = "2d2h.computernetwork.clc.fitus@gmail.com"** và **password = "jvys wjpg mmmx belp"**: Tên người dùng và mật khẩu của tài khoản Gmail được sử dụng để đăng nhập và truy cập email.
- + **imap = imaplib.IMAP4_SSL("imap.gmail.com")**: Tạo một kết nối SSL với máy chủ IMAP của Gmail để lấy email.
- + **imap.login(username, password)**: Xác thực với tên người dùng và mật khẩu đã cung cấp.
- + **imap.select("inbox")**: Chọn hộp thư "inbox" để thao tác.
- + **status, email_ids = imap.search(None, 'UNSEEN')**: Tìm kiếm các email chưa đọc trong hộp thư.
- + **if not email_ids[0]: ...**: Kiểm tra xem có email chưa đọc nào không. Nếu không tìm thấy, in ra thông báo "No unread emails found." và trả về chuỗi "Error".

- + **latest_email_id = email_ids[0].split()[-1]**: Chọn ID của email cuối cùng trong danh sách các email chưa đọc.
 - + **status, email_data = imap.fetch(latest_email_id, "(RFC822)")**: Lấy dữ liệu của email cuối cùng theo ID.
 - + **sender_email, body = extract_email_information(email_data[0][1])**: Trích xuất thông tin về người gửi và nội dung email từ dữ liệu email đã lấy.
 - + **command = process_email(sender_email, body)**: Gọi hàm **process_email()** để xử lý thông tin từ email như các yêu cầu được đưa ra trong nội dung email và trả về lệnh cần thực hiện.
 - + **imap.logout()**: Đóng kết nối với máy chủ IMAP, đồng thời thoát khỏi tài khoản email.
 - + **return command**: Trả về lệnh đã nhận được sau khi xử lý email
- **if __name__ == "__main__": ...**: Được sử dụng để kiểm tra xem mã Python đang được thực thi trực tiếp từ chương trình chính hay không. Nếu đoạn mã được chạy trực tiếp (không phải là một module được nhập vào từ một chương trình khác), thì các dòng code bên dưới **if __name__ == "__main__":** sẽ được thực thi. Trong trường hợp này, hàm **read_mail()** sẽ được gọi.

3. sendMail.py

```

1 SMTP_HOST = 'smtp.gmail.com'
2 SMTP_USER = '2d2h.computernetwork.clc.fitus@gmail.com'
3 SMTP_PASS = 'jvys wjpg mmmx belp'
4 FROM_EMAIL = '2d2h.computernetwork.clc.fitus@gmail.com' # Update the from email address

```

- + **SMTP_HOST = 'smtp.gmail.com'**: Đây là địa chỉ máy chủ SMTP của Gmail. Khi gửi email, phải kết nối đến máy chủ này để chuyển thư đi.
- + **SMTP_USER = '2d2h.computernetwork.clc.fitus@gmail.com'**: Đây là địa chỉ email của người gửi, sẽ được sử dụng để đăng nhập vào máy chủ SMTP.
- + **SMTP_PASS = 'jvys wjpg mmmx belp'**: Đây là mật khẩu của địa chỉ email người gửi, được sử dụng để xác thực khi đăng nhập vào máy chủ SMTP. Lưu ý rằng việc lưu mật khẩu trực tiếp trong mã nguồn không được khuyến khích do

có thể tạo ra rủi ro bảo mật. Trong môi trường thực tế, nên sử dụng biện pháp bảo mật cao hơn, chẳng hạn như lưu mật khẩu trong biến môi trường hoặc sử dụng các dịch vụ quản lý mật khẩu.

- + **FROM_EMAIL = '2d2h.computernetwork.clc.fitus@gmail.com'**: Đây là địa chỉ email người gửi. Nếu cần thay đổi địa chỉ email người gửi, bạn có thể chỉnh sửa giá trị của biến này.

- ***sendMail:***

```
1 def send_email(subject, body, recipient_email, attachment_path=None):
2     msg = MIMEMultipart()
3     msg['From'] = FROM_EMAIL
4     msg['To'] = recipient_email
5     msg['Subject'] = subject
6
7     # Attach text content
8     text = MIMEText(body, 'plain')
9     msg.attach(text)
10
11     # Attach the screenshot or any other attachment
12     if attachment_path:
13         with open(attachment_path, 'rb') as attachment_file:
14             attachment = MIMEImage(attachment_file.read(), name='screenshot.png')
15             msg.attach(attachment)
16
17     # Connect, authenticate, and send mail
18     smtp_server = SMTP_SSL(SMTP_HOST, port=SMTP_SSL_PORT)
19     smtp_server.set_debuglevel(1) # Show SMTP server interactions
20     smtp_server.login(SMTP_USER, SMTP_PASS)
21     smtp_server.sendmail(FROM_EMAIL, recipient_email, msg.as_string())
22
23     # Disconnect
24     smtp_server.quit()
```

- + **msg = MIMEMultipart()**: Tạo một đối tượng MIMEMultipart, là một đối tượng MIME sử dụng để xây dựng các thông điệp email với nhiều phần khác nhau.
- + **msg['From'] = FROM_EMAIL**: Thiết lập trường "From" của thông điệp email bằng địa chỉ email người gửi (**FROM_EMAIL**).
- + **msg['To'] = recipient_email**: Thiết lập trường "To" của thông điệp email bằng địa chỉ email của người nhận (**recipient_email**).
- + **msg['Subject'] = subject**: Thiết lập trường "Subject" của thông điệp email bằng tiêu đề được truyền vào hàm (**subject**).


- + **text = MIMEText(body, 'plain')**: Tạo một đối tượng MIMEText chứa nội dung văn bản của email (**body**) với định dạng 'plain'.
- + **msg.attach(text)**: Đính kèm đối tượng MIMEText vào đối tượng MIMEMultipart để thêm nội dung văn bản vào thông điệp email.
- + **if attachment_path::** Kiểm tra xem có đường dẫn đính kèm (**attachment_path**) không.
- + **with open(attachment_path, 'rb') as attachment_file::** Mở tệp đính kèm ở chế độ đọc dưới dạng nhị phân ('rb').
- + **attachment = MIMEImage(attachment_file.read(), name='screenshot.png')**: Tạo một đối tượng MIMEImage từ dữ liệu của tệp đính kèm, và đặt tên tệp là 'screenshot.png'.
- + **msg.attach(attachment)**: Đính kèm đối tượng MIMEImage vào đối tượng MIMEMultipart để thêm hình ảnh hoặc bất kỳ đính kèm nào khác vào thông điệp email.
- + **smtp_server = SMTP_SSL(SMTP_HOST, port=SMTP_SSL_PORT)**: Tạo một đối tượng SMTP_SSL để kết nối đến máy chủ SMTP của Gmail qua SSL.
- + **smtp_server.set_debuglevel(1)**: Thiết lập chế độ debug cho đối tượng SMTP để hiển thị tất cả tương tác với máy chủ SMTP.
- + **smtp_server.login(SMTP_USER, SMTP_PASS)**: Đăng nhập vào máy chủ SMTP bằng tên người dùng và mật khẩu.
- + **smtp_server.sendmail(FROM_EMAIL, recipient_email, msg.as_string())**: Gửi thông điệp email đến người nhận.
- + **smtp_server.quit()**: Ngắt kết nối từ máy chủ SMTP sau khi đã gửi email xong.

- *send_screenshot_email:*

```
1 def send_screenshot_email(recipient_email):
2     screenshot_path = 'Assets/screenshot.png'
3     screenshot = ImageGrab.grab()
4     screenshot.save(screenshot_path)
5
6     subject = "SCREENSHOT"
7     body = "This is the screenshot!"
8
9     send_email(subject, body, recipient_email, screenshot_path)
10
```

- + **screenshot_path = 'Assets/screenshot.png'**: Xác định đường dẫn nơi mà ảnh chụp màn hình sẽ được lưu trữ.
- + **screenshot = ImageGrab.grab()**: Sử dụng hàm **grab()** từ thư viện PIL để chụp màn hình và lưu vào biến **screenshot**.
- + **screenshot.save(screenshot_path)**: Lưu ảnh chụp màn hình vào đường dẫn đã xác định ở bước 1.
- + **subject = "SCREENSHOT"**: Đặt chủ đề của email là "SCREENSHOT".
- + **body = "This is the screenshot!"**: Thiết lập nội dung văn bản của email là "This is the screenshot!".
- + **send_email(subject, body, recipient_email, screenshot_path)**: Gọi hàm **send_email** để gửi email với chủ đề, nội dung và ảnh chụp màn hình đã được xác định.

- ***send_process_email:***



```
1 def send_process_email(recipient_email):
2     app_controller = AppController()
3     formatted_table = app_controller.viewList()
4
5     subject = "LIST OF APPLICATIONS"
6     body = formatted_table # Use the formatted table as the email body
7
8     send_email(subject, body, recipient_email)
```

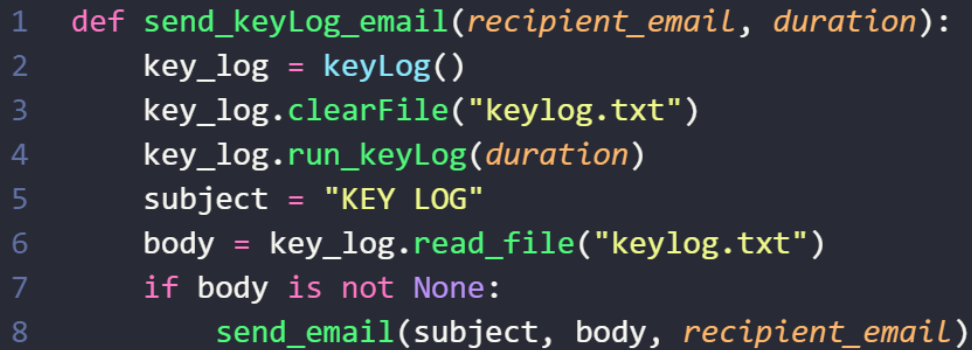
- + **app_controller = AppController()**: Tạo một đối tượng của lớp **AppController**. Lớp này chứa các phương thức để quản lý app trên hệ thống.
- + **formatted_table = app_controller.viewList()**: Gọi phương thức **viewList** từ đối tượng **app_controller** để lấy danh sách các app và định dạng nó thành một bảng.
- + **subject = "LIST OF APPLICATIONS"**: Đặt chủ đề của email là "LIST OF APPLICATIONS".
- + **body = formatted_table**: Sử dụng bảng đã được định dạng như là nội dung email.
- + **send_email(subject, body, recipient_email)**: Gọi hàm **send_email** để gửi email với chủ đề, nội dung và không có tệp đính kèm (ảnh chụp màn hình, v.v.) cho địa chỉ email đã chỉ định.

- *send_bgProcess_email:*



```
1 def send_bgProcess_email(recipient_email):
2     proc_controller = ProcessController()
3     formatted_table = proc_controller.viewList()
4
5     subject = "LIST OF BACKGROUND PROCESSES"
6     body = formatted_table # Use formatted table as the email body
7
8     send_email(subject, body, recipient_email)
```

- + **proc_controller = ProcessController()**: Tạo một đối tượng của lớp **ProcessController**. Lớp này chứa các phương thức để quản lý các process nền trên hệ thống.
 - + **formatted_table = proc_controller.viewList()**: Gọi phương thức **viewList** từ đối tượng **proc_controller** để lấy danh sách các process nền và định dạng nó thành một bảng.
 - + **subject = "LIST OF BACKGROUND PROCESSES"**: Đặt chủ đề của email là "LIST OF BACKGROUND PROCESSES".
 - + **body = formatted_table**: Sử dụng bảng đã được định dạng như là nội dung email.
 - + **send_email(subject, body, recipient_email)**: Gọi hàm **send_email** để gửi email với chủ đề, nội dung và không có tệp đính kèm (ảnh chụp màn hình, v.v.) cho địa chỉ email đã chỉ định.
-
- **send_keyLog_email**:



```
1 def send_keyLog_email(recipient_email, duration):
2     key_log = keyLog()
3     key_log.clearFile("keylog.txt")
4     key_log.run_keyLog(duration)
5     subject = "KEY LOG"
6     body = key_log.read_file("keylog.txt")
7     if body is not None:
8         send_email(subject, body, recipient_email)
```

- + **key_log = keyLog()**: Tạo một đối tượng của lớp **keyLog**. Lớp này chứa các phương thức để ghi và đọc dữ liệu.
- + **key_log.clearFile("keylog.txt")**: Gọi phương thức **clearFile** của đối tượng **key_log** để xóa nội dung của tệp "keylog.txt". Điều này giúp đảm bảo rằng chỉ có thông tin mới nhất được lưu trữ.
- + **key_log.run_keyLog(duration)**: Gọi phương thức **run_keyLog** để bắt đầu ghi lại các sự kiện từ bàn phím trong một khoảng thời gian **duration**.
- + **subject = "KEY LOG"**: Đặt chủ đề của email là "KEY LOG".
- + **body = key_log.read_file("keylog.txt")**: Gọi phương thức **read_file** của đối tượng **key_log** để đọc nội dung từ tệp "keylog.txt" và lưu trữ nó trong biến **body**.
- + **if body is not None**:: Kiểm tra xem **body** có giá trị hay không.
- + **send_email(subject, body, recipient_email)**: Gọi hàm **send_email** để gửi email với chủ đề "KEY LOG" và nội dung là nội dung đã đọc từ tệp "keylog.txt" tới địa chỉ email đã chỉ định.

- **send_startApp:**




```
1 def send_startApp(recipient_email, appName):  
2     app_controller = ApplicationController()  
3  
4     subject = "APPLICATION STATUS REPORT"  
5     body = app_controller.startApp(appName)  
6     send_email(subject, body, recipient_email)
```

- + **app_controller = ApplicationController():** Tạo một đối tượng của lớp **AppController**. Lớp này chứa các phương thức để quản lý và kiểm soát các app trên hệ thống.
 - + **subject = "APPLICATION STATUS REPORT":** Đặt chủ đề của email là "APPLICATION STATUS REPORT".
 - + **body = app_controller.startApp(appName):** Gọi phương thức **startApp** của đối tượng **app_controller** với **appName** làm đối số để lấy thông tin về việc khởi động một app và lưu trữ thông tin này trong biến **body**.
 - + **send_email(subject, body, recipient_email):** Gọi hàm **send_email** để gửi email với chủ đề "APPLICATION STATUS REPORT" và nội dung là thông tin về việc khởi động một app tới địa chỉ email đã chỉ định.
-
- **send_endApp:**



```
1  def send_endApp(recipient_email, app_name):  
2      app_controller = AppController()  
3  
4      subject = "APPLICATION STATUS REPORT"  
5      body = app_controller.endApp(app_name)  
6      send_email(subject, body, recipient_email)
```

- + **app_controller = AppController()**: Tạo một đối tượng của lớp **AppController**. Lớp này chứa các phương thức để quản lý và kiểm soát các app trên hệ thống.
 - + **subject = "APPLICATION STATUS REPORT"**: Đặt chủ đề của email là "APPLICATION STATUS REPORT".
 - + **body = app_controller.endApp(app_name)**: Gọi phương thức **endApp** của đối tượng **app_controller** với **app_name** làm đối số để lấy thông tin về việc kết thúc một app và lưu trữ thông tin này trong biến **body**.
 - + **send_email(subject, body, recipient_email)**: Gọi hàm **send_email** để gửi email với chủ đề "APPLICATION STATUS REPORT" và nội dung là thông tin về việc kết thúc một app tới địa chỉ email đã chỉ định.
-
- *send_startProc_status:*




```

1 def send_startProc_status(recipient_email, process_name):
2     proc_controlller = ProcessController()
3
4     subject = "PROCESS STATUS REPORT"
5     body = proc_controlller.startBackgroundProcess(process_name)
6     send_email(subject, body, recipient_email)

```

- + **proc_controller = ProcessController()**: Tạo một đối tượng của lớp **ProcessController**. Lớp này chứa các phương thức để quản lý và kiểm soát các process trên hệ thống.
- + **subject = "PROCESS STATUS REPORT"**: Đặt chủ đề của email là "PROCESS STATUS REPORT".
- + **body = proc_controller.startBackgroundProcess(process_name)**: Gọi phương thức **startBackgroundProcess** của đối tượng **proc_controller** với **process_name** làm đối số để lấy thông tin về việc bắt đầu một process nền và lưu trữ thông tin này trong biến **body**.
- + **send_email(subject, body, recipient_email)**: Gọi hàm **send_email** để gửi email với chủ đề "PROCESS STATUS REPORT" và nội dung là thông tin về việc bắt đầu một process nền tới địa chỉ email đã chỉ định.

- ***send_endProc_status:***



```

1 def send_endProc_status(recipient_email, process_name):
2     proc_controlller = ProcessController()
3
4     subject = "PROCESS STATUS REPORT"
5     body = proc_controlller.endProcess(process_name)
6     send_email(subject, body, recipient_email)


```

- + **proc_controller = ProcessController()**: Tạo một đối tượng của lớp **ProcessController**. Lớp này chứa các phương thức để quản lý và kiểm soát các process trên hệ thống.
- + **subject = "PROCESS STATUS REPORT"**: Đặt chủ đề của email là "PROCESS STATUS REPORT".
- + **body = proc_controller.endProcess(process_name)**: Gọi phương thức **endProcess** của đối tượng **proc_controller** với **process_name** làm đối số để lấy thông tin về việc kết thúc một process và lưu trữ thông tin này trong biến **body**.
- + **send_email(subject, body, recipient_email)**: Gọi hàm **send_email** để gửi email với chủ đề "PROCESS STATUS REPORT" và nội dung là thông tin về việc kết thúc một process tới địa chỉ email đã chỉ định.

4. keyLog.py

Class **keyLog()**: bao gồm các hàm liên quan đến ghi bàn phím:

- *clearFile*:



```

1  def clearFile(self, file_path):
2      with open(file_path, 'w') as file:
3          file.truncate(0)

```


- + **with open(file_path, 'w') as file**: Mở tệp đường dẫn được chỉ định trong chế độ ghi ('w'). Sử dụng từ khóa **with** giúp đảm bảo tệp sẽ được đóng lại sau khi hoàn thành, ngay cả khi có lỗi xảy ra.
- + **file.truncate(0)**: Gọi phương thức **truncate** với tham số là 0. Phương thức này cắt độ dài của tệp xuống còn 0, tức là xóa tất cả nội dung của tệp.

- *on_key_event*:

- + **def on_key_event()**: Định nghĩa một hàm tên là **on_key_event** nhận vào hai tham số: **key** và **is_press**.

- + **try:** Bắt đầu một khối try để bắt các ngoại lệ có thể xảy ra.
- + **if is_press:** Kiểm tra nếu sự kiện là nhấn phím.
- + **with open("keylog.txt", "a", encoding='utf-8') as f:** Mở tệp "keylog.txt" trong chế độ ghi tiếp ('a') với mã hóa UTF-8. Tệp này sẽ được tạo nếu nó không tồn tại.
- + **f.write('Key pressed: {0}\n'.format(key)):** Ghi chuỗi 'Key pressed: {0}\n' vào tệp, với {0} được thay thế bằng giá trị của key. Điều này ghi lại phím đã được nhấn.
- + **except AttributeError:** Bắt ngoại lệ AttributeError, có thể xảy ra khi cố gắng truy cập một thuộc tính không tồn tại của đối tượng key.
- + **pass:** Nếu có AttributeError, hàm sẽ không làm gì và tiếp tục thực thi. Điều này giúp xử lý các phím đặc biệt mà không gây ra lỗi.

- *read_file:*



```

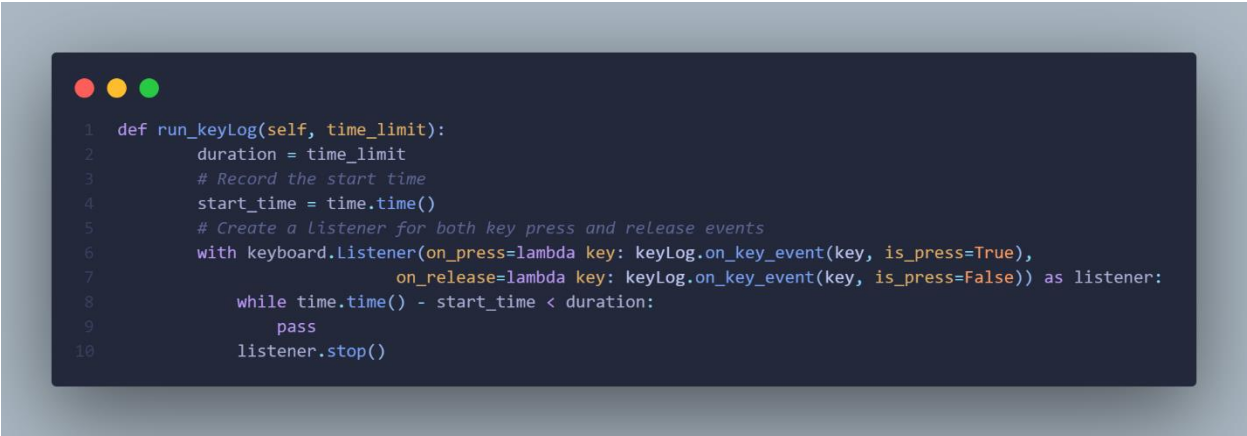
1  def read_file(self, file_path):
2      try:
3          with open(file_path, 'r') as file:
4              body = file.read()
5              return body
6      except FileNotFoundError:
7          print(f"File not found: {file_path}")
8          return None

```

- + **def read_file(self, file_path):** Định nghĩa một hàm read_file nhận vào một tham số file_path.
- + **try:** Bắt đầu một khối try để bắt các ngoại lệ có thể xảy ra.
- + **with open(file_path, 'r') as file:** Mở tệp đường dẫn được chỉ định trong chế độ đọc ('r'). Sử dụng từ khóa with giúp đảm bảo tệp sẽ được đóng lại sau khi hoàn thành, ngay cả khi có lỗi xảy ra.
- + **body = file.read():** Đọc toàn bộ nội dung của tệp và gán nó vào biến body.
- + **return body:** Trả về nội dung của tệp.
- + **except FileNotFoundError:** Bắt ngoại lệ FileNotFoundError, có thể xảy ra khi tệp không tồn tại.

- + **print(f'File not found: {file_path}')**: In ra thông báo lỗi với đường dẫn tệp không tìm thấy.
- + **return None**: Trả về None khi không tìm thấy tệp.

- **run_keyLog:**



```

1  def run_keyLog(self, time_limit):
2      duration = time_limit
3      # Record the start time
4      start_time = time.time()
5      # Create a listener for both key press and release events
6      with keyboard.Listener(on_press=lambda key: keyLog.on_key_event(key, is_press=True),
7                             on_release=lambda key: keyLog.on_key_event(key, is_press=False)) as listener:
8          while time.time() - start_time < duration:
9              pass
10         listener.stop()


```

- + **def run_keyLog(self, time_limit):** Định nghĩa một hàm run_keyLog nhận vào tham số time_limit.
- + **duration = time_limit**: Đặt thời gian chạy cho keylogger.
- + **start_time = time.time()**: Ghi lại thời gian bắt đầu.
- + **with keyboard.Listener(...) as listener**: Tạo một listener để theo dõi cả sự kiện nhấn phím và thả phím.
- + **on_press=lambda key: keyLog.on_key_event(key, is_press=True)**: Khi một phím được nhấn, hàm on_key_event sẽ được gọi với tham số is_press được đặt là True.
- + **on_release=lambda key: keyLog.on_key_event(key, is_press=False)**: Khi một phím được thả, hàm on_key_event sẽ được gọi với tham số is_press được đặt là False.
- + **while time.time() - start_time < duration**: Vòng lặp sẽ tiếp tục cho đến khi thời gian hiện tại trừ đi thời gian bắt đầu vượt quá thời gian chạy đã định.
- + **listener.stop()**: Sau khi vòng lặp kết thúc, listener sẽ được dừng lại.

5. appController.py

Class **AppController()**: bao gồm các hàm liên quan đến điều khiển app:

- **process2List:**



```

1  def process2List(self, processes):
2      a = processes.decode().strip()
3      b = a.split("\r\n")
4      b = [" ".join(x.split()) for x in b]
5      c = [x.split() for x in b][2:]
6      return c

```

- + **a = processes.decode().strip()**: Chuyển đổi chuỗi byte thành chuỗi văn bản thông qua phương thức **decode()**, sau đó loại bỏ các khoảng trắng ở đầu và cuối chuỗi bằng phương thức **strip()**. Kết quả là chuỗi văn bản đã được chuẩn hóa. Giải mã chuỗi byte thành chuỗi thông thường, loại bỏ khoảng trắng dẫn/dẫn cuối.
- + **b = a.split("\r\n")**: Tách chuỗi thành một danh sách các dòng, sử dụng ký tự xuống dòng ("**\r\n**") làm điểm tách. Tách mỗi dòng thành một danh sách các từ.
- + **b = [" ".join(x.split()) for x in b]**: Cho mỗi dòng trong danh sách, loại bỏ các khoảng trắng thừa ở giữa bằng cách sử dụng **split()** để tách từng từ trong dòng, sau đó **join()** để kết hợp lại thành một chuỗi mới. Kết quả là danh sách các dòng đã được chuẩn hóa.
- + **c = [x.split() for x in b][2:]**: Cho mỗi dòng trong danh sách sau bước trước, tách chuỗi thành các phần tử và lưu thành danh sách con. Bỏ qua hai phần tử đầu tiên của danh sách (hàng tiêu đề không cần thiết).
- + **return c**: Trả về danh sách cuối cùng, đại diện cho thông tin chi tiết về các tiến trình, được biểu diễn dưới dạng danh sách các danh sách.

- ***viewList***:



```

1  def viewList(self):
2      app = subprocess.check_output(
3          "powershell gps | where {$_.MainWindowTitle} | \
4          select Name,Id,@{Name='ThreadCount';Expression={$_Threads.Count}}",
5          stdin=subprocess.PIPE, stderr=subprocess.PIPE)
6
7      self.appList = self.process2List(app)
8
9      # Format the appList as a table
10     table_headers = ["Process", "PID", "Thread Count"]
11     formatted_table = tabulate(self.appList, headers=table_headers, tablefmt="pretty")
12
13     return formatted_table

```

- + **subprocess.check_output**: Là một hàm trong thư viện **subprocess** của Python, được sử dụng để thực thi một lệnh hệ thống và trả về kết quả của lệnh đó dưới dạng chuỗi byte.
- + **"powershell gps | where {\$_.MainWindowTitle} | select Name,Id,@{Name='ThreadCount';Expression={\$_Threads.Count}}"**
 - **powershell gps**: Chạy cmdlet gps trong PowerShell để lấy thông tin về các tiến trình (Get-Process).
 - **where {\$_.MainWindowTitle}**: Lọc các tiến trình dựa trên có cửa sổ chính hay không.
 - **select Name,Id,@{Name='ThreadCount';Expression={\$_Threads.Count}}**: Chọn các thuộc tính Name, Id và tạo một thuộc tính mới là ThreadCount, thể hiện số lượng luồng cho mỗi tiến trình.
- + Kết quả của lệnh PowerShell được trả về dưới dạng chuỗi byte và được lưu vào biến **app**
- + **self.appList = self.process2List(app)**: Kết quả được truyền vào hàm process2List để chuyển đổi từ chuỗi byte sang danh sách các tiến trình.
- + **table_headers = ["Process", "PID", "Thread Count"]**: Định nghĩa tiêu đề cho bảng dữ liệu.
- + **formatted_table = tabulate(self.appList, headers=table_headers, tablefmt="pretty")**: Sử dụng thư viện **tabulate** để định dạng danh sách các tiến trình thành một bảng dễ đọc. Kết quả được trả về dưới dạng một chuỗi chứa bảng được định dạng.
- + **return formatted_table**: Trả về bảng dữ liệu đã được định dạng.

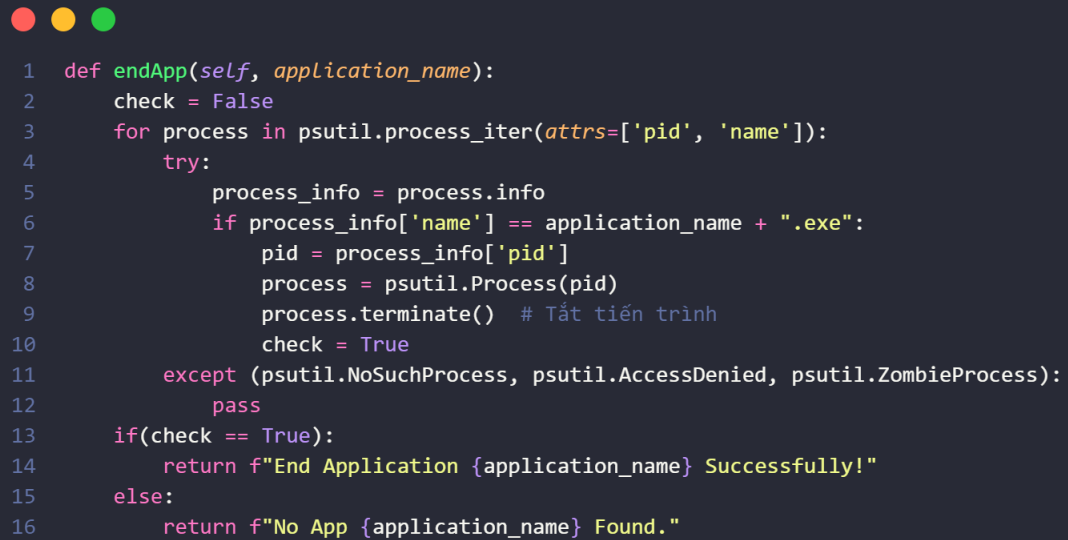
- **startApp**:



```
1 def startApp(self, application_name):
2     try:
3         PathProcess = os.path.realpath(application_name + ".exe")
4         os.startfile(PathProcess)
5         return f"Started application: {application_name}"
6     except Exception as e:
7         return f"Error starting application: {e}"
```

- + **PathProcess = os.path.realpath(application_name + ".exe")**: Xây dựng đường dẫn tương đối đến tập tin thực thi của app. Chuỗi ".exe" được thêm vào tên app để đảm bảo rằng extension của tập tin thực thi đã được xác định.
- + **os.startfile(PathProcess)**: Sử dụng hàm **startfile** từ module **os** để mở tập tin hoặc app với chương trình mặc định được liên kết với loại tập tin đó trên hệ điều hành. Trong trường hợp này, nó sẽ mở app có đường dẫn là **PathProcess**.
- + **return f"Started application: {application_name}"**: Trả về một chuỗi thông báo thành công khi app được khởi động. Chuỗi này chứa tên app để người dùng biết app nào đã được bắt đầu.
- + **except Exception as e: return f"Error starting application: {e}"**: Nếu có bất kỳ lỗi nào xảy ra trong quá trình khởi động app, chương trình sẽ rơi vào khối **except**. Nó sẽ trả về một chuỗi thông báo lỗi chứa thông tin chi tiết về lỗi đó, giúp người dùng xác định vấn đề

- **endApp**:



```

1  def endApp(self, application_name):
2      check = False
3      for process in psutil.process_iter(attrs=['pid', 'name']):
4          try:
5              process_info = process.info
6              if process_info['name'] == application_name + ".exe":
7                  pid = process_info['pid']
8                  process = psutil.Process(pid)
9                  process.terminate() # Tắt tiến trình
10                 check = True
11             except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
12                 pass
13         if(check == True):
14             return f"End Application {application_name} Successfully!"
15         else:
16             return f"No App {application_name} Found."

```

- + **check = False**: Tạo một biến **check** và khởi tạo giá trị là **False**. Biến này được sử dụng để kiểm tra xem có tìm thấy và kết thúc app hay không.
- + **for process in psutil.process_iter(attrs=['pid', 'name'])**:: Sử dụng vòng lặp để duyệt qua tất cả các tiến trình đang chạy trên hệ thống sử dụng **psutil.process_iter**. Thông tin về tiến trình được thu thập bao gồm **pid** (Process ID) và **name** (tên tiến trình).
- + Trong **try**, **process_info = process.info** lấy thông tin về tiến trình.
- + **if process_info['name'] == application_name + ".exe"**:: Kiểm tra xem tên của tiến trình có khớp với tên app được đưa vào hay không.
- + **pid = process_info['pid']**: Lấy Process ID (PID) của tiến trình.
- + **process = psutil.Process(pid)**: Tạo một đối tượng **Process** từ PID để thao tác với tiến trình.
- + **process.terminate()**: Gọi phương thức **terminate()** để kết thúc tiến trình.
- + **check = True**: Đặt giá trị của biến **check** thành **True** để cho biết tiến trình đã được kết thúc thành công.
- + **except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess)**: Bắt các ngoại lệ có thể xảy ra trong quá trình lấy thông tin về tiến trình, như khi không tìm thấy tiến trình (**NoSuchProcess**), truy cập bị từ chối (**AccessDenied**), hoặc tiến trình là Zombie (**ZombieProcess**).
- + **if(check == True): return f"End Application {application_name} Successfully!"**: Kiểm tra giá trị của **check**. Nếu **check** là **True**, thì trả về một chuỗi thông báo cho biết app đã được kết thúc thành công.

- + **else: return f"No App {application_name} Found.":** Nếu **check** là **False**, thì trả về một chuỗi thông báo cho biết không tìm thấy app với tên đã đưa vào.

6. processController.py

Class **ProcessController()**: chứa các hàm dùng để tương tác với process, bao gồm:

- **process2List:**

```
1 def process2List(self, processes):
2     a = processes.decode().strip()
3     b = a.split("\r\n")
4     b = [" ".join(x.split()) for x in b]
5     c = [x.split() for x in b][2:]
6
7     # Wrap the first column to a maximum width of 200 characters
8     max_width = 200
9     for item in c:
10         item[0] = '\n'.join([item[0][i:i+max_width] for i in range(0, len(item[0]), max_width)])
11
12     return c
```

- + Giống như trong **AppController**, hàm **process2List** của **ProcessController** nhận đầu vào là kết quả của lệnh PowerShell được gọi bằng **subprocess.check_output**.
- + Kết quả này được chuyển thành một danh sách các process với thông tin như tên process, ID và số lượng luồng.
- + Do tên của process có thể rất dài, đối với mỗi process, cột đầu tiên (tên process) được chia thành các dòng có chiều dài tối đa là 200 ký tự.

- **viewList:**

```
1 def viewList(self):
2     app = subprocess.check_output(
3         "powershell Get-Process | Where-Object { $_.MainWindowTitle.Length -eq 0 } | Select-Object Name,Id,@(Name='ThreadCount';Expression={$_.Threads.Count})",
4         stdin=subprocess.PIPE, stderr=subprocess.PIPE)
5
6     self.applist = self.process2List(app)
7
8     # Format the applist as a table
9     table_headers = ["Process", "PID", "Thread Count"]
10    formatted_table = tabulate(self.applist, headers=table_headers, colAlign=("left", "center", "center"), tablefmt="pretty",)
11
12    return formatted_table
```

- + **viewList** sử dụng **subprocess.check_output** để chạy lệnh PowerShell, lấy danh sách các process không có cửa sổ chính (**Where-Object { \$_.MainWindowTitle.Length -eq 0 }**).
- + Sau đó, gọi **process2List** để chuyển đổi kết quả thành danh sách và sử dụng **tabulate** để định dạng danh sách này thành một bảng.

- ***startBackgroundProcess:***


```

1 def startBackgroundProcess(self, process_name):
2     try:
3         subprocess.Popen(process_name, shell=True, stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
4         return f"Started process: {process_name}"
5     except Exception as e:
6         return f"Error starting process: {e}"

```

- + Sử dụng **subprocess.Popen** để bắt đầu một process mới dựa trên tên process được truyền vào:
 - **shell=True**: Đặt giá trị này thành True cho phép sử dụng PowerShell để thực hiện câu lệnh.
 - **stdin=subprocess.PIPE**: Thiết lập luồng đầu vào (stdin) của tiến trình. Trong trường hợp này, **subprocess.PIPE** được sử dụng để tạo một ống dẫn đầu vào từ quy trình Python chính đến tiến trình con.
 - **stdout=subprocess.PIPE**: Thiết lập luồng đầu ra (stdout) của tiến trình. Tương tự như stdin, **subprocess.PIPE** tạo một ống dẫn đầu ra từ tiến trình con đến quy trình Python chính.
 - **stderr=subprocess.PIPE**: Thiết lập luồng lỗi (stderr) của tiến trình. Cũng giống như stdin và stdout, **subprocess.PIPE** tạo một ống dẫn đến quy trình Python chính.
- + Nếu có lỗi, hàm sẽ trả về một thông báo lỗi.

- ***endProcess:***



```

1  def endProcess(self, process_name):
2      check = False
3      for process in psutil.process_iter(attrs=['pid', 'name']):
4          try:
5              process_info = process.info
6              if process_info['name'] == process_name:
7                  pid = process_info['pid']
8                  process = psutil.Process(pid)
9                  process.terminate()
10                 check = True
11             except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
12                 pass
13         if(check == True):
14             return (f"End Process {process_name} Successfully!")
15         else:
16             return (f"No Process {process_name} Found.")

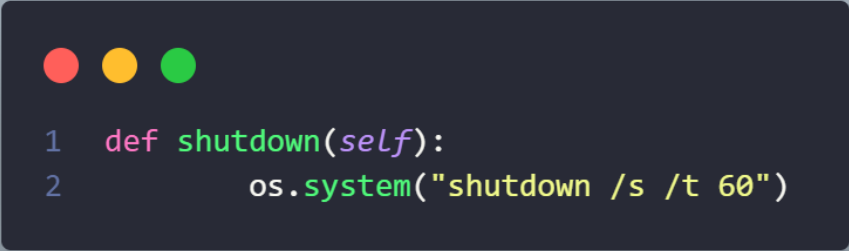
```

- + **check = False**: Tạo một biến **check** và khởi tạo giá trị là **False**. Biến này được sử dụng để kiểm tra xem có tìm thấy và kết thúc app hay không.
- + **for process in psutil.process_iter(attrs=['pid', 'name'])**:: Sử dụng vòng lặp để duyệt qua tất cả các process đang chạy trên hệ thống sử dụng **psutil.process_iter**. Thông tin về process được thu thập bao gồm **pid** (Process ID) và **name** (tên tiến trình).
- + Trong **try**, **process_info = process.info** lấy thông tin về tiến trình.
- + **if process_info['name'] == process_name**:: Kiểm tra xem tên của process có khớp với tên process được đưa vào hay không.
- + **pid = process_info['pid']**: Lấy Process ID (PID) của tiến trình.
- + **process = psutil.Process(pid)**: Tạo một đối tượng **Process** từ PID để thao tác.
- + **process.terminate()**: Gọi phương thức **terminate()** để kết thúc tiến trình.
- + **check = True**: Đặt giá trị của biến **check** thành **True** để cho biết tiến trình đã được kết thúc thành công.
- + **except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess)**: Bắt các ngoại lệ có thể xảy ra trong quá trình lấy thông tin về process, như khi không tìm thấy (**NoSuchProcess**), truy cập bị từ chối (**AccessDenied**), hoặc process là Zombie (**ZombieProcess**).

- + **if(check == True): return f"End Process {process_name} Successfully!":** Kiểm tra giá trị của **check**. Nếu **check** là **True**, thì trả về một chuỗi thông báo cho biết process đã được kết thúc thành công.
- + **else: return f"No Process {process_name} Found.":** Nếu **check** là **False**, thì trả về một chuỗi thông báo cho biết không tìm thấy process với tên đã đưa vào.

7. powerController.py

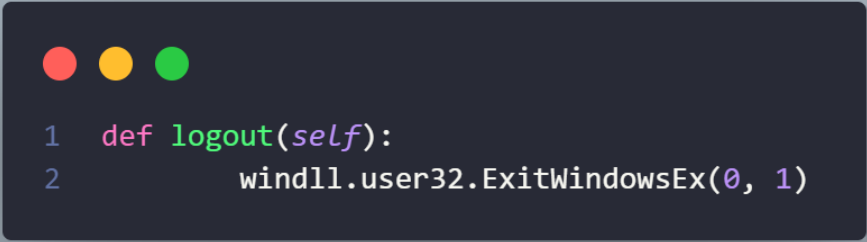
- **shutdown:**



```
1 def shutdown(self):
2     os.system("shutdown /s /t 60")
```

- + Sử dụng hàm **os.system** để thực thi lệnh **shutdown /s /t 60**, tắt máy tính sau 60 giây.

- **logout:**



```
1 def logout(self):
2     windll.user32.ExitWindowsEx(0, 1)
```

- + Sử dụng hàm **ExitWindowsEx** từ **windll.user32** để đăng xuất người dùng khỏi hệ thống. Tham số **0** đại diện cho đăng xuất, và **1** đại diện cho lựa chọn đóng tất cả các ứng dụng trước khi đăng xuất.

V. Hướng dẫn sử dụng

1. Thiết lập ứng dụng điều khiển từ xa

a) Thiết bị được điều khiển

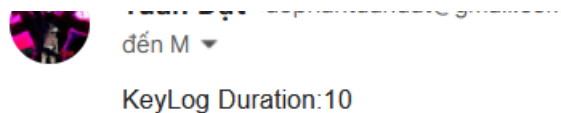
- Vào folder **source** → **dist** → **main**
- Chạy file **main.exe**.
- Nhập thời gian muốn dò đọc email (nhập số tự nhiên, số có thể âm hoặc dương, nếu là số âm thì chương trình sẽ chạy vĩnh viễn)
- Nhấn vào nút *Start*, chương trình sẽ bắt đầu đọc mail.

b) Thiết bị điều khiển

- Không cần thiết lập. Chỉ cần gửi email cho 2d2h.computernetwork.clc.fitus@gmail.com
- Có thể gửi từ bất kỳ thiết bị nào
- Mail không nhất thiết phải có tiêu đề. Trong nội dung mail sẽ có những từ khóa để kích hoạt tính năng nhất định
- Từ khóa và các yêu cầu phải nằm trong một dòng riêng biệt trong nội dung thư. Mọi nội dung khác của email sẽ được bỏ qua

2. Ghi bàn phím

- Gửi mail có chứa dòng lệnh: **KeyLog Duration:<thời gian (giây)>**
- VD: **Keylog Duration:10** để ghi bàn phím trong 10 giây kể từ khi chương trình đọc được mail



- Sau khi hết thời hạn ghi bàn phím, người gửi sẽ nhận được reply là nội dung đã ghi được



đến tôi ▾



Tiếng Anh ▾

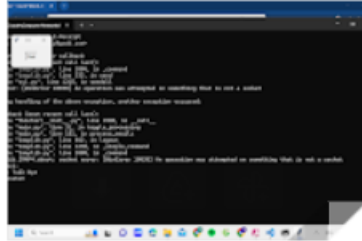


Tiếng Việt ▾

[Dịch thư](#)

This is the screenshot!

Một tệp đính kèm • Gmail đã quét ⓘ



↩ Trả lời

➡ Chuyển tiếp

4. Lấy danh sách các app đang hoạt động

- Gửi email có chứa dòng lệnh: **ListApp**



đến M ▾

ListApp

↩ Trả lời

➡ Chuyển tiếp

- Sau một lúc, người gửi sẽ nhận được email có nội dung là danh sách các app đang hoạt động trên thiết bị được điều khiển



đến tôi ▾



Tiếng Anh ▾



Tiếng Việt ▾

[Dịch thư](#)

Process	PID	Thread Count
ApplicationFrameHost	11432	4
chrome	13160	41
Code	11344	48
HxOutlook	1688	32
SystemSettings	10288	44
TextInputHost	6476	19

↶ Trả lời

↷ Chuyển tiếp

5. Lấy danh sách các process đang hoạt động

- Gửi email có chứa dòng lệnh: **ListProcess**



đến M ▾

ListProcess

↶ Trả lời

↷ Chuyển tiếp

- Sau một lúc, người gửi sẽ nhận được email rất dài có nội dung là danh sách các process đang hoạt động trên thiết bị được điều khiển



đến tôi ▾



Tiếng Anh ▾



Tiếng Việt ▾

[Dịch thư](#)

Process	PID	Thread Count
AggregatorHost	5920	4
AGMService	4732	4
armsvc	4676	2
audiodg	8136	7
chrome	1420	14
chrome	1776	10
chrome	1796	23
chrome	2816	25
chrome	5464	13
chrome	5792	20
chrome	6836	8
chrome	7744	25
chrome	8352	23
chrome	9136	23
chrome	10156	28
chrome	11236	10
chrome	11568	24
chrome	12072	23
chrome	12568	14

6. Khởi động app

- Gửi email chứa dòng lệnh: **StartApp Name =<tên app>**
- Tên app không có “.exe”. Ví dụ:



đến 2d2h.computernetwork.clc.fitus ▾

StartApp Name =Notepad

← Trả lời

→ Chuyển tiếp

- Sau một lúc, người nhận sẽ nhận được email thông báo đã thành công hay thất bại trong việc thực thi lệnh



2d2h.computernetwork.clc.fitus@gmail.com

đến tôi ▾



Tiếng Anh ▾



Tiếng Việt ▾

[Dịch thư](#)

Started application: Notepad

↩ Trả lời

➡ Chuyển tiếp

7. Tắt app

- Gửi email chứa dòng lệnh: **EndApp Name =<tên app>**
- Tên app cách dấu bằng một khoảng trắng, không có “.exe”. Ví dụ:



đến 2d2h.computernetwork.clc.fitus ▾

EndApp Name =Notepad

↩ Trả lời

➡ Chuyển tiếp

- Sau một lúc, người nhận sẽ nhận được email thông báo đã thành công hay thất bại trong việc thực thi lệnh



2d2h.computernetwork.clc.fitus@gmail.com

đến tôi ▾



Tiếng Anh ▾



Tiếng Việt ▾

[Dịch thư](#)

End Application Notepad Successfully!

↩ Trả lời

➡ Chuyển tiếp

8. Khởi động process

- Gửi email chứa dòng lệnh: **StartProcess Name =<tên process>**
- Tên process bắt buộc phải có “.exe”. Ví dụ:



đến 2d2h.computernetwork.clc.fitus ▾

StartProcess Name =explorer.exe

↩ Trả lời

➡ Chuyển tiếp

- Sau một lúc, người nhận sẽ nhận được email thông báo đã thành công hay thất bại trong việc thực thi lệnh



2d2h.computernetwork.clc.fitus@gmail.com

đến tôi ▾



Tiếng Anh ▾



Tiếng Việt ▾

[Dịch thư](#)

Started process: explorer.exe

↩ Trả lời

➡ Chuyển tiếp

9. Tắt process

- Gửi email chứa dòng lệnh: **EndProcess Name =<tên process>**
- Tên process bắt buộc phải có “.exe”. Ví dụ:



đến 2d2h.computernetwork.clc.fitus ▾

EndProcess Name =explorer.exe

↩ Trả lời

➡ Chuyển tiếp

- Sau một lúc, người nhận sẽ nhận được email thông báo đã thành công hay thất bại trong việc thực thi lệnh



2d2h.computernetwork.clc.fitus@gmail.com

đến tôi ▾



Tiếng Anh ▾



Tiếng Việt ▾

[Dịch thư](#)

End Process explorer.exe Successfully!

↶ Trả lời

↷ Chuyển tiếp

* Lưu ý: explorer.exe là tên của File Explorer, đồng thời còn là process quan trọng của hệ điều hành Windows, không nên bật tắt bừa bãi

10. Đăng xuất khỏi Windows

- Gửi email có dòng lệnh: **Log out**
- Sau khi đọc được lệnh, thiết bị được điều khiển sẽ đăng xuất ra khỏi Windows
- Sẽ không có email trả lời kết quả thực thi lệnh cho người gửi

11. Tắt nguồn

- Gửi email có dòng lệnh: **Shut Down**
- Sau khi đọc được lệnh, thiết bị được điều khiển sẽ đếm ngược 1 phút trước khi tắt nguồn
- Sẽ không có email trả lời kết quả thực thi lệnh cho người gửi

12. Tắt ứng dụng điều khiển từ xa

c) Cách 1: Tắt trên thiết bị được điều khiển

- Chỉ cần tắt file .exe là được

d) Cách 2: Tắt bằng email

- Gửi email có dòng lệnh: **Die**
- Sau khi đọc được lệnh, chương trình sẽ ngừng đọc mail, mặc dù file .exe vẫn đang chạy
- Sẽ không có email trả lời kết quả thực thi lệnh cho người gửi

VI. Đóng góp

Thành viên	MSSV	Công việc	Tỉ lệ (%) hoàn thành
Trần Nguyễn Minh Hoàng	22127131	<ul style="list-style-type: none">- Tính năng đọc email, key log, screenshot, thiết kế GUI trong source code.- Các phần giải thích code trong báo cáo tương ứng với các tính năng đã làm.- Quay video, edit video hướng dẫn sử dụng.	100%
Lê Hồ Phi Hoàng	22127123	<ul style="list-style-type: none">- Tính năng bật app, tắt app trong source code.- Các phần giải thích code trong báo cáo tương ứng với các tính năng đã làm.- Quay video hướng dẫn sử dụng.	100%
Phạm Thành Đạt	22127064	<ul style="list-style-type: none">- Tính năng soạn email, liệt kê app, tắt nguồn và đăng xuất Windows trong source code.- Các phần giải thích code trong báo cáo tương ứng với các tính năng đã làm.- Quay video hướng dẫn sử dụng.	100%
Đỗ Phan Tuấn Đạt	22127057	<ul style="list-style-type: none">- Tính năng liệt kê process, bật process, tắt process trong source code.- Các phần giải thích code trong báo cáo tương ứng với các tính năng đã làm.- Viết hướng dẫn sử dụng trong báo cáo, tổng hợp và thiết kế báo cáo	100%

VII. Tài liệu tham khảo

- Đọc email bằng python: <https://thepythoncode.com/article/reading-emails-in-python>
- Sử dụng module subprocess: <https://www.dataquest.io/blog/python-subprocess/>
- Gửi email bằng python: https://www.tutorialspoint.com/python/python_sending_email.htm

- Cách ghi lại bàn phím bằng python: <https://www.youtube.com/watch?v=mDY3v2Xx-Q4&t=163s>
- Chụp ảnh màn hình bằng python: <https://nitratine.net/blog/post/how-to-take-a-screenshot-in-python-using-pil/>
- Tắt máy tính bằng python: <https://www.geeksforgeeks.org/python-script-to-shutdown-computer/>