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

ĐỒ ÁN LÍ THUYẾT GIỮA KỲ MẠNG MÁY TÍNH

ĐỒ ÁN: REMOTE CONTROL

Giảng viên lý thuyết: Thầy Đỗ Hoàng Cường
Lớp: 21CTT2
Nhóm thực hiện: 14

Thành Phố Hồ Chí Minh, tháng 12 năm 2022

MỤC LỤC

A. Thông tin đồ án

I. Thông tin thành viên

Bảng thông tin thành viên trong đồ án:

MÃ SỐ SINH VIÊN	HỌ VÀ TÊN SINH VIÊN
21120058	Phạm Nhật Duy
21120102	Nguyễn Trúc Nguyên
21120146	Lê Nguyễn Phương Thùy
21120150	Nguyễn Song Toàn
21120158	Trương Công Trung
21120179	Nguyễn Đặng Đăng Khoa

II. Phân chia công việc và mức độ hoàn thành

Bảng thông tin thành viên trong đồ án:

CÔNG VIỆC	NGƯỜI THỰC HIỆN	MỨC ĐỘ HOÀN THÀNH
Giao diện	21120150	100%
Application	21120150	100%
Process	21120102	100%
Screenshot	21120146	100%
Keylogger	21120058 + 21120158	100%
Shutdown	21120179	100%
Quay video	21120058 + 21120179	100%

III. Ý tưởng chính của đồ án

- Xác định kiến trúc mạng**: Client – Server**
- Giao thức sử dụng tầng Transport**: TCP (Transmission Control Protocol)**
  Giao thức TCP cung cấp các dịch vụ truyền thông cho các ứng dụng mạng: truyền dữ liệu tin cậy. TCP cho phép các ứng dụng trên các máy chủ được nối mạng có thể tạo các "kết nối" với nhau, mà qua đó chúng có thể trao đổi dữ liệu hoặc các gói tin. Giao thức này đảm bảo dữ liệu gửi từ bên gửi đến bên nhận chính xác và đúng thứ tự.
- Port được sử dụng: **8888**
- Vì số kết nối giữa client và server là 1 kết nối, nên khi thực hiện những chức năng mà có hiển thị ra form để thực hiện, thì khi thực hiện xong, sẽ tắt form đó để thực hiện tiếp những chức năng khác. Nếu không tắt form, sẽ dẫn đến lỗi lặp.

IV. Môi trường lập trình và các thư viện ngoài hệ thống

- Phần mềm biên soạn và biên dịch ngôn ngữ lập trình Java: Netbeans
- Thư viện hỗ trợ cho chức năng Keylogger: ***jnativehook-2.1.0.jar***

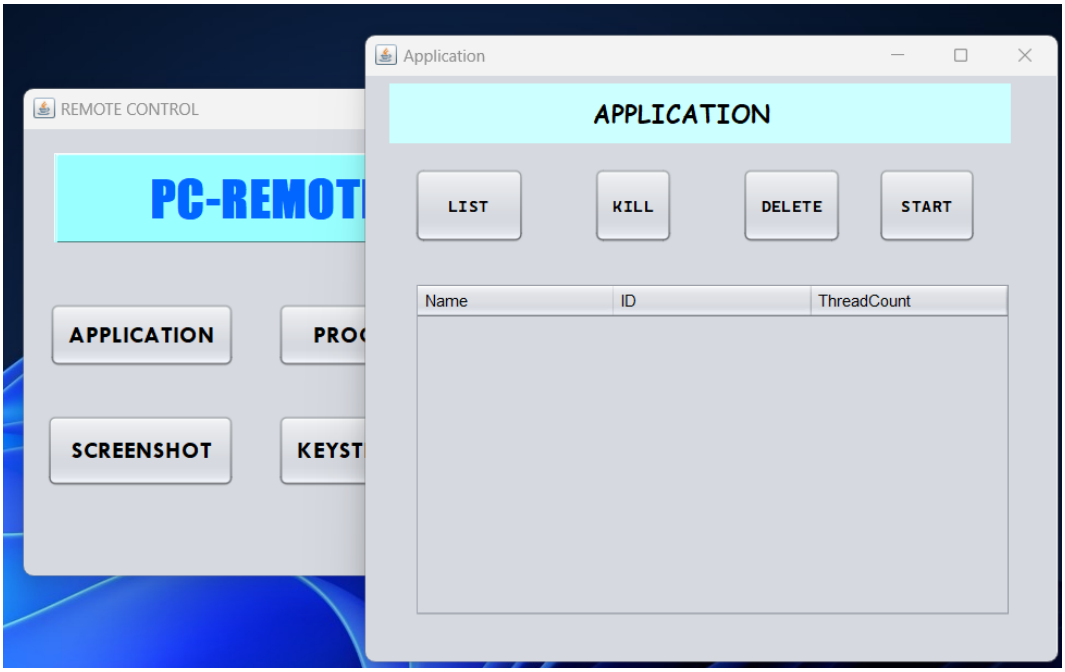
B. Mô tả chức năng của các hàm, thủ tục (chia làm 2 project: Client và Server)

I. Project: Client

Package: client gồm các class:

1. Application (*Application.java*)

Chức năng: Hiển thị form gồm các nút bấm để xử lí, gửi tín hiệu đến server để thực hiện các chức năng liên quan đến application: list running apps, kill app, start app và xóa các dòng trong bảng list running apps. Gửi request qua server để thực hiện các chức năng và nhận response từ server.



Các hàm, thủ tục:

+ Khai báo các biến cSocket, dOut, dIn và tạo Constructor Application với tham chiếu cSocket:

```
13      Socket cSocket;
14      DataOutputStream dOut;
15      DataInputStream dIn;
16
17      public Application(Socket cSocket) throws IOException {
18          initComponents();
19          this.cSocket = cSocket;
20          this.dOut = new DataOutputStream( out:cSocket.getOutputStream());
21          this.dIn = new DataInputStream( in:cSocket.getInputStream());
22
23      }
```

+ Hàm *sendSignal(String s, Socket sc)* với tham chiếu s, sc dùng để ghi dữ liệu để gửi request cho server thông qua biến *dOut*.

+ Hàm *jButton1ActionPerformed()* được thực hiện khi nhấn nút LIST, lúc này sẽ gửi tín hiệu "LISTAPP" qua server thông qua hàm sendSignal("LISTAPP", cSocket), nhận số lượng dòng kết quả khi chạy lệnh cmd từ server lưu vào biến apps, sau đó xử lí từng dòng và tách chuỗi để được 3 biến: name, id, countThread.

Sau đó, ghi từng dòng với 3 cột: name, id, countThread vào Table để hiển thị ra cửa sổ Application.

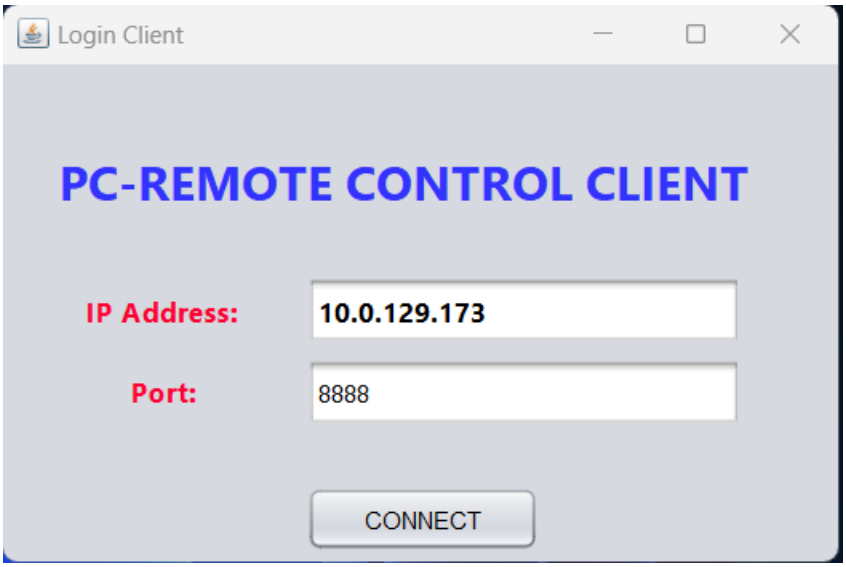
+ Hàm *jButton2ActionPerformed()* được thực hiện khi nhấn nút KILL, lúc này sẽ gửi tín hiệu "KILLAPP" qua server thông qua hàm sendSignal("KILLAPP", cSocket) và hiển thị form của class KILL.

+ Hàm *jButton3ActionPerformed()* được thực hiện khi nhấn nút DELETE, lúc này xóa hết các dòng của Table ở cửa sổ Application.

+ Hàm *jButton4ActionPerformed()* được thực hiện khi nhấn nút START, lúc này sẽ gửi tín hiệu "STARTAPP" qua server thông qua hàm sendSignal("STARTAPP", cSocket) và hiển thị form của class START.

2. ClientFrame (*ClientFrame.java*)

Chức năng: Hiển thị form kết nối đến server và xử lí vấn đề kết nối đến server.



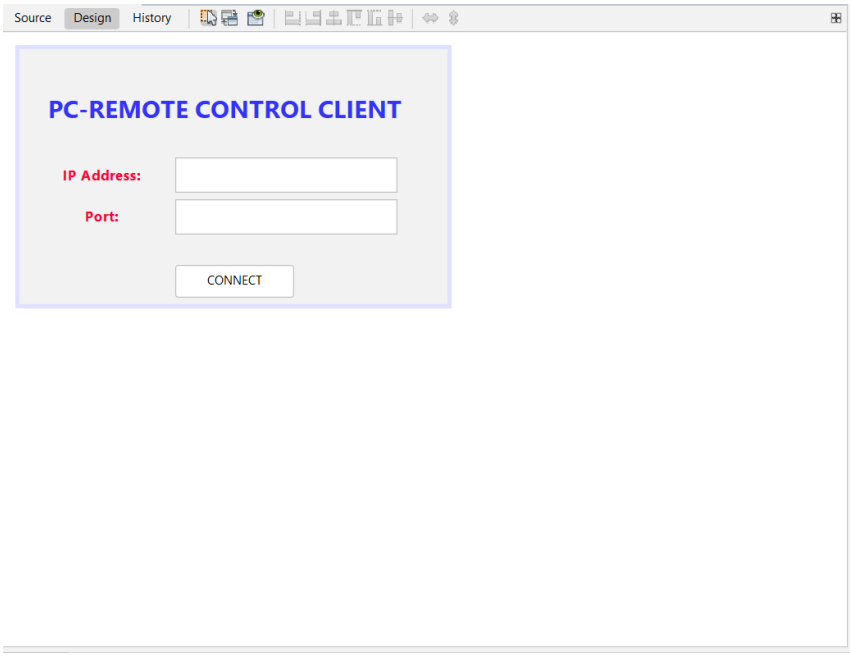
Các hàm, thủ tục:

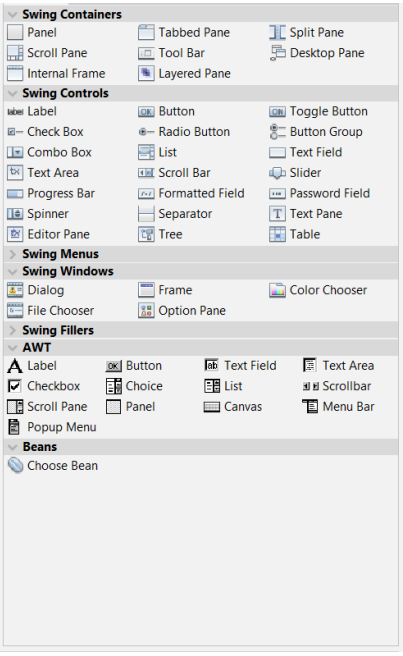
+ Khai báo các biến để sử dụng cho toàn bộ class

```
17      Socket sc = null;
18      JOptionPane jOptionPane = new JOptionPane();
19      String ip, port, message;
```

+ Hàm initComponents() là hàm do hệ thống Netbeans tạo ra khi người lập trình thực hiện tạo form trên phần Design.

```
32      private void initComponents() {
33
34          jLabel1 = new javax.swing.JLabel();
35          jLabel2 = new javax.swing.JLabel();
36          IPField = new javax.swing.JTextField();
37          jLabel3 = new javax.swing.JLabel();
38          PortField = new javax.swing.JTextField();
39          jButton1 = new javax.swing.JButton();
40
41          setDefaultCloseOperation( operation: javax.swing.WindowConstants.EXIT_ON_CLOSE);
42          setTitle( title: "Login Client");
43
44          jLabel1.setFont(new java.awt.Font( name: "Segoe UI",  style: 1,  size: 24)); // NOI18N
45          jLabel1.setForeground(new java.awt.Color( r: 51,  g: 51,  b: 255));
46          jLabel1.setHorizontalAlignment( alignment: javax.swing.SwingConstants.CENTER);
47          jLabel1.setText( text: "PC-REMOTE CONTROL CLIENT");
48
49          jLabel2.setFont(new java.awt.Font( name: "Segoe UI",  style: 1,  size: 14)); // NOI18N
50          jLabel2.setForeground(new java.awt.Color( r: 255,  g: 0,  b: 51));
51          jLabel2.setHorizontalAlignment( alignment: javax.swing.SwingConstants.CENTER);
52          jLabel2.setText( text: "IP Address:");
53
54          IPField.setFont(new java.awt.Font( name: "Segoe UI",  style: 1,  size: 14)); // NOI18N
55
56          jLabel3.setFont(new java.awt.Font( name: "Segoe UI",  style: 1,  size: 14)); // NOI18N
57          jLabel3.setForeground(new java.awt.Color( r: 255,  g: 0,  b: 51));
58          jLabel3.setHorizontalAlignment( alignment: javax.swing.SwingConstants.CENTER);
59          jLabel3.setText( text: "Port:");
60
61          jButton1.setText( text: "CONNECT");
```

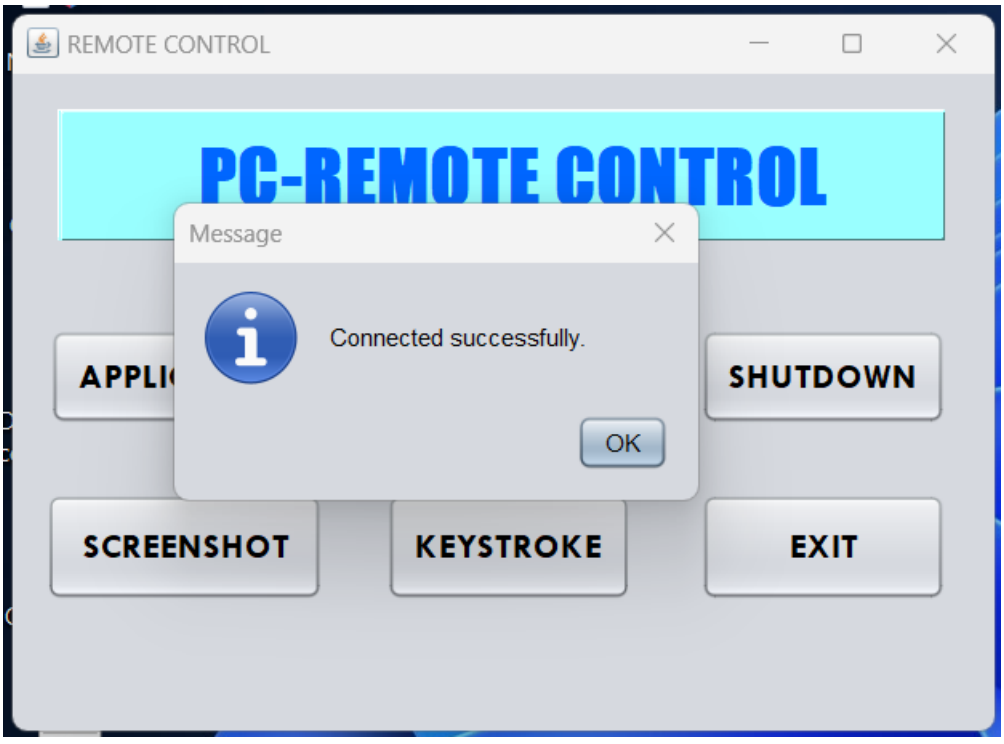




+ Hàm jButton1ActionPerformed() được thực hiện khi nhấn vào nút bấm Connect ở Login Client form. Hàm này để kiểm tra IP và port có đúng không, nếu đúng thì

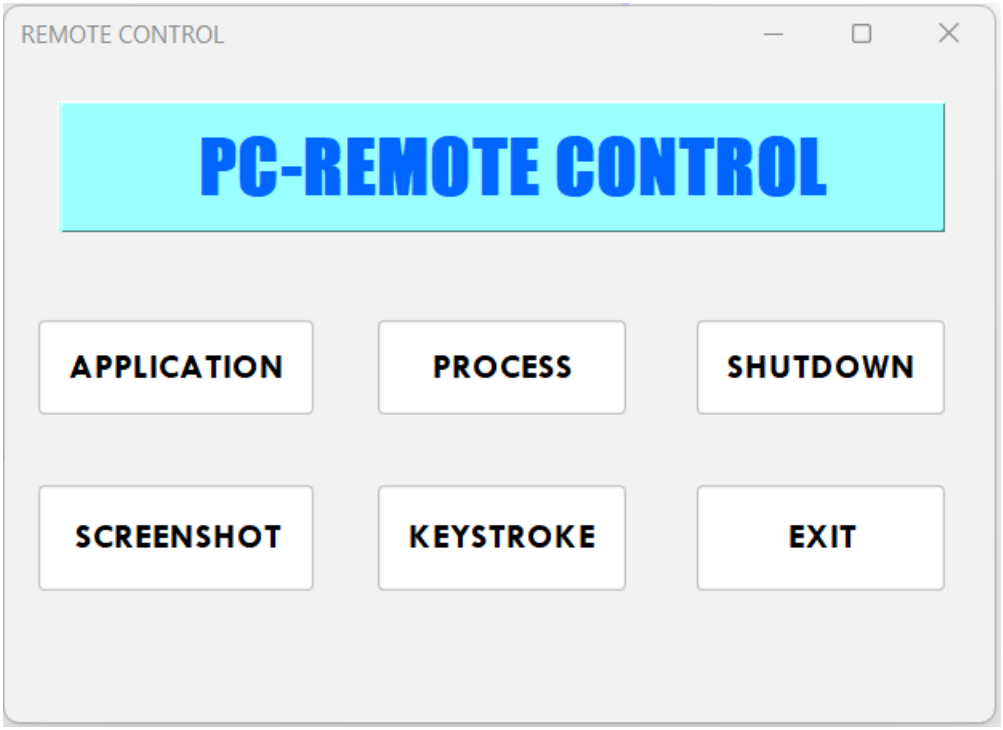
thực hiện kết nối với server và gọi class ClientHandler, hiển thị form của class ClientHandler và hiển thị sẽ hiển thị MessageDialog Connected successfully.

```
109     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
110         // TODO add your handling code here:
111         ip = IPField.getText().trim();
112         port = PortField.getText().trim();
113         boolean check = true;
114 
115         if (ip.isEmpty() || port.isEmpty()) {
116             message = "Please enter all values!";
117         } else {
118                 try {
119                     sc = new Socket( host: IPField.getText(),  port: Integer.parseInt( s: PortField.getText()));
120                     ClientHandler ch = new ClientHandler( cSocket: sc);
121                     ch.setVisible( b: true);
122                     this.dispose();
123                 } catch (UnknownHostException e) {
124                     e.printStackTrace();
125                     message = "Error in connecting.";
126                     check = false;
127                 } catch (IOException e) {
128                     e.printStackTrace();
129                     message = "Error in connecting.";
130                     check = false;
131                 }
132         }
133 
134         if (check == true) {
135             message = "Connected successfully.";
136         }
137         JOptionPane.showMessageDialog( parentComponent: this, message);
138     }
```



3. ClientHandler (*ClientHandler.java*)

Chức năng: Hiển thị form gồm các nút bấm thực hiện các chức năng lớn: APP, PROCESS, SHUTDOWN, KEYSTROKE (KEYLOGGER), SCREENSHOT và thoát chương trình. Gửi request qua server và nhận response từ server.



Các hàm, thủ tục:

+ Khai báo các biến cần dùng cho class: cSocket (Kiểu Socket), DataInputStream *dIn* (dùng để đọc dữ liệu từ luồng bên ngoài vào), DataOutputStream *dOut* (để ghi dữ liệu mà sau này có thể được đọc bởi một DataInputStream)

+ Tạo constructor để sử dụng qua các class khác với dữ liệu đầu vào là Socket

```
19      Socket cSocket;
20      DataOutputStream dOut;
21      DataInputStream dIn;
22
23      public ClientHandler(Socket cSocket) throws IOException {
24          initComponents();
25          this.cSocket = cSocket;
26          this.dOut = new DataOutputStream( out:cSocket.getOutputStream());
27          this.dIn = new DataInputStream( in:cSocket.getInputStream());
28      }
```

+ Hàm *sendSignal(String s, Socket sc)* với tham chiếu s, sc dùng để ghi dữ liệu để gửi request cho server thông qua biến *dOut*.

+ Hàm *jButton1ActionPerformed()* được thực hiện khi nhấn nút APPLICATION, lúc này sẽ gửi tín hiệu "APP" qua server thông qua hàm sendSignal("APP", cSocket) và hiển thị form của class Application.

+ Hàm *jButton2ActionPerformed()* được thực hiện khi nhấn nút PROCESS, lúc này sẽ gửi tín hiệu "PROCESS" qua server thông qua hàm sendSignal("PROCESS", cSocket) và hiển thị form của class Process.
+ Hàm *jButton3ActionPerformed()* được thực hiện khi nhấn nút KEYSTROKE, lúc này sẽ gửi tín hiệu "KEYLOGGER" qua server thông qua hàm sendSignal("KEYLOGGER", cSocket) và hiển thị form của class Keylogger.
+ Hàm *jButton4ActionPerformed()* được thực hiện khi nhấn nút SCREENSHOT, lúc này sẽ gửi tín hiệu "SCREEN" qua server thông qua hàm sendSignal("SCREEN", cSocket) và hiển thị form của class Screen.
+ Hàm *jButton5ActionPerformed()* được thực hiện khi nhấn nút EXIT, lúc này sẽ gửi tín hiệu "EXIT" qua server thông qua hàm sendSignal("EXIT", cSocket) và đóng kết nối của client, tắt cửa sổ chương trình
+ Hàm *jButton6ActionPerformed()* được thực hiện khi nhấn nút SHUTDOWN, lúc này sẽ gửi tín hiệu "SHUTDOWN" qua server thông qua hàm sendSignal("SHUTDOWN", cSocket).
+ Hàm *formWindowClosed()* được thực hiện khi nhận nút X của cửa sổ chương trình. Hàm này sẽ sẽ gửi tín hiệu "EXIT" qua server thông qua hàm sendSignal("EXIT", cSocket) và đóng kết nối của client, tắt cửa sổ chương trình.

4. KILL (*KILL.java*)

Chức năng: Hiển thị form để điền ID của app hoặc process cần kill và thực hiện gửi request đến server để Kill app hoặc process bằng ID, nhận response từ server.
Các hàm, thủ tục:
+ Khai báo các biến cSocket, dIn, dOut và tạo constructor KILL

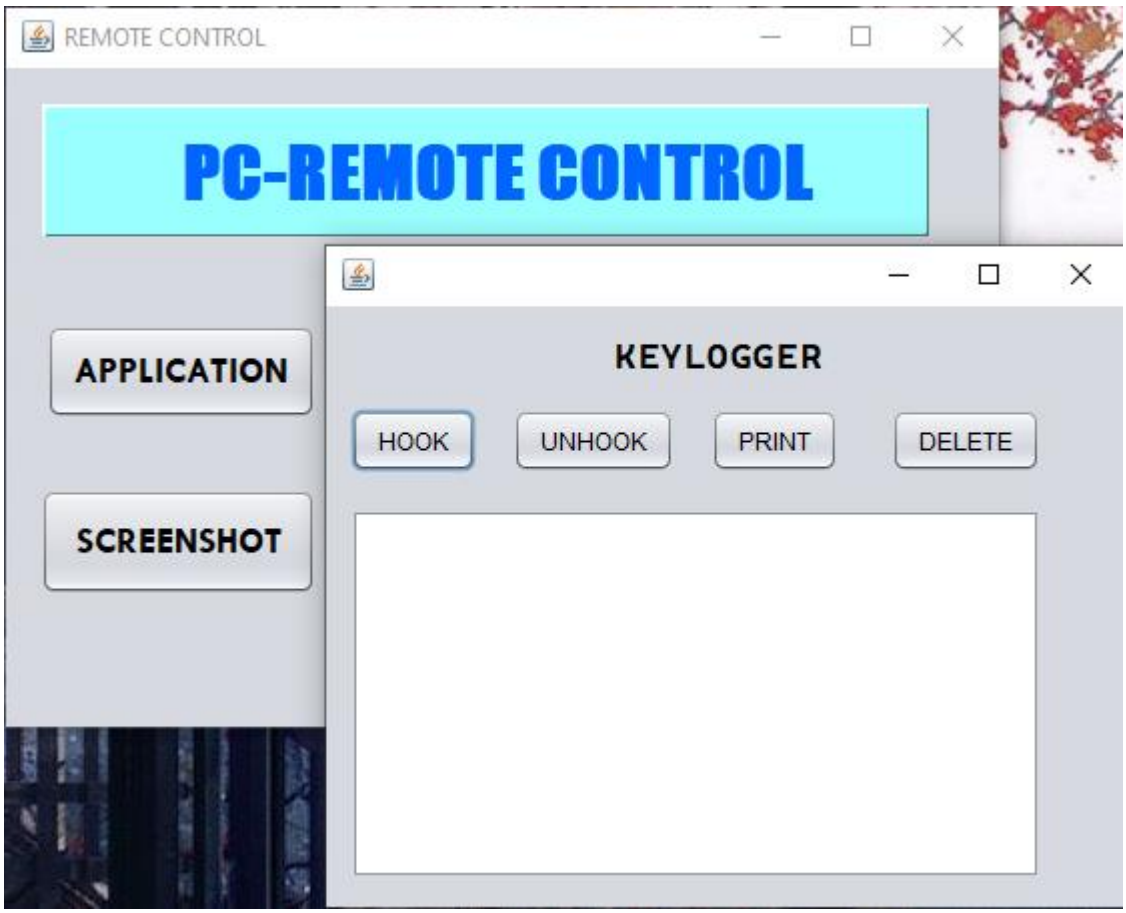
```
16      Socket cSocket;
17      DataOutputStream dOut;
18      DataInputStream dIn;
19      JOptionPane jOptionPane = new JOptionPane();
20
21      public KILL(Socket cSocket) throws IOException {
22          initComponents();
23          this.cSocket = cSocket;
24          this.dOut = new DataOutputStream( out: cSocket.getOutputStream());
25          this.dIn = new DataInputStream( in: cSocket.getInputStream());
26      }
```

+ Hàm *jButton1ActionPerformed()* được thực hiện khi nhấn nút KILL, lấy ID từ textField của người dùng nhập vào lưu vào biến *string s* ở Form KILL, gửi string s đến server qua *dOut*, server xử lý và sau đó nhận dữ liệu từ server gửi về qau *dIn*, nếu nhận byte 1 thì hiển thị MessageDialog "Kill successfully!", ngược lại thì hiển thị MessageDialog "Kill unsuccessfully!".
+ Hàm *formWindowClosed()* được thực hiện khi nhấn nút đóng cửa sổ KILL. Hàm này gửi tín hiệu "EXITKILL" đến server qua dOut và đóng cửa sổ form KILL.

5. Keylogger (*Keylogger.java*)

Chức năng:

Hiển thị form gồm các nút bấm để xử lí, gửi tín hiệu đến server để thực hiện các chức năng liên quan đến Key logger : bắt đầu bắt phím của máy Server (TAKE), dừng việc bắt phím ở máy server, in các phím đã bắt được ra màn hình và xóa màn hình . Gửi request qua server để thực hiện các chức năng và nhận response từ server.



Các hàm, thủ tục:

+ Khai báo các biến cSocket, dOut, dIn, img và tạo Constructor Keylogger với tham chiếu cSocket:

```
12      Socket cSocket;
13      DataOutputStream dOut;
14      DataInputStream dIn;
15
16      public Keylogger(Socket cSocket) throws IOException {
17          initComponents();
18          this.cSocket = cSocket;
19          this.dOut = new DataOutputStream( out: cSocket.getOutputStream());
20          this.dIn = new DataInputStream( in: cSocket.getInputStream());
21      }
```

+ Hàm sendSignal(String s, Socket sc) với tham chiếu s, sc dùng để ghi dữ liệu để gửi request cho server thông qua biến dOut.

```
125     public void sendSignal(String s, Socket sc) throws IOException {
126         dOut.writeUTF( str: s);
127         dOut.flush();
128     }
```

+ Hàm jButton1ActionPerformed() được thực hiện khi nhấn nút HOOK. Lúc này sẽ gửi tín hiệu "HOOK" qua server thông qua hàm sendSignal("HOOK", cSocket). Khi nhận request từ client, server sẽ tiến hành bắt phím nhấn và lưu vào một mảng được khởi tạo trước. Vì thế có thể bắt được 1 lần rất nhiều phím.

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
140     try {
141         sendSignal( s: "HOOK", sc: cSocket);
142     } catch (IOException ex) {
143         Logger.getLogger( name: Keylogger.class.getName()).log( level: Level.SEVERE,  msg: null,  thrown: ex);
144     }
145 }
```

```
349     if (aRe.equals( anObject: "HOOK")) {
350         // Get the logger for "org.jnativehook" and set the level to warning.
351         java.util.logging.Logger logger = java.util.logging.Logger.getLogger( name: GlobalScreen.class.getPackage().getName());
352         logger.setLevel( newLevel: java.util.logging.Level.WARNING);
353         // Disable the parent handlers
354         logger.setUseParentHandlers( useParentHandlers: false);
355
356         GlobalScreen.registerNativeHook(); // Register the native hook to the system to listen to the keyboard
357         GlobalScreen.addNativeKeyListener(new ServerHandler()); // Add the native key listener to the system
358     }
```

ServerHandler.java

+ Hàm jButton2ActionPerformed() được thực hiện khi nhấn nút UNHOOK. Lúc này sẽ gửi tín hiệu "UNHOOK" qua server thông qua hàm sendSignal("UNHOOK",cSocket). Khi nhận request từ client, server sẽ dừng việc bắt phím nhấn lại.

```
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
148     try {
149         sendSignal( s: "UNHOOK", sc: cSocket);
150     } catch (IOException ex) {
151         Logger.getLogger( name: Keylogger.class.getName()).log( level: Level.SEVERE,  msg: null,  thrown: ex);
152     }
153 }
```

```
360             if (aRe.equals( anObject: "UNHOOK")) {
361                 GlobalScreen.unregisterNativeHook();
362             }
```

+ Hàm jButton3ActionPerformed() được thực hiện khi nhấn nút PRINT. Lúc này sẽ gửi tín hiệu "PRINT" qua server thông qua hàm sendSignal("PRINT",cSocket). Server nhận request "PRINT" từ client thông qua và sẽ trả về client mảng các phím đã bắt được rồi in ra màn hình

```
    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
156         try {
157             sendSignal( s: "PRINT", sc: cSocket);
158             String key = dIn.readUTF();
159             System.out.println( x: key);
160             jTextArea1.setText( t: "");
161             jTextArea1.append(key + "\n");
162 
163         } catch (IOException ex) {
164             Logger.getLogger( name: Keylogger.class.getName()).log( level: Level.SEVERE,  msg: null,  thrown: ex);
165         }
166     }
```

```
364             if (aRe.equals( anObject: "PRINT")) {
365                 //System.out.println("Result : " + result);
366                 dOut.writeUTF( str: result);
367                 dOut.flush();
368             }
```

+ Hàm jButton4ActionPerformed() được thực hiện khi nhấn nút DELETE. Lúc này sẽ gửi tín hiệu "DELETE" qua server thông qua hàm sendSignal("DELETE",cSocket). Khi đó mảng các nút bấm bắt được được in ra trên màn hình sẽ bị xóa.

6. Process (*Process.java*)

Chức năng: Hiển thị form gồm các nút bấm để xử lí, gửi tín hiệu đến server để thực hiện các chức năng liên quan đến process: list process, kill process, start process và xóa các dòng trong bảng list process. Gửi request qua server để thực hiện các chức năng và nhận response từ server.

Các hàm, thủ tục:

+ Khai báo các biến cSocket, dOut, dIn và tạo Constructor Process với tham chiếu cSocket:

```
11  public class Process extends javax.swing.JFrame {
12
13      Socket cSocket;
14      DataOutputStream dOut;
15      DataInputStream dIn;
16
17      public Process(Socket cSocket) throws IOException {
18          initComponents();
19          this.cSocket = cSocket;
20          this.dOut = new DataOutputStream( out: cSocket.getOutputStream());
21          this.dIn = new DataInputStream( in: cSocket.getInputStream());
22      }
```

+ Hàm *sendSignal(String s, Socket sc)* với tham chiếu s, sc dùng để ghi dữ liệu để gửi request cho server thông qua biến *dOut*.

```
24      public void sendSignal(String s, Socket sc) throws IOException {
25          dOut.writeUTF( str: s);
26          dOut.flush();
27      }
```

+ Hàm *jButton1ActionPerformed()* được thực hiện khi nhấn nút LIST, lúc này sẽ gửi tín hiệu "LISTPRO" qua server thông qua hàm sendSignal("LISTPRO", cSocket), nhận số lượng dòng kết quả khi chạy lệnh cmd từ server lưu vào biến pros, sau đó xử lí từng dòng và tách chuỗi để được 5 biến: name, id, sessionName, session#. Sau đó, ghi từng dòng với 5 cột trên vào Table để hiển thị ra cửa sổ Process.

+ Hàm *jButton2ActionPerformed()* được thực hiện khi nhấn nút KILL, lúc này sẽ gửi tín hiệu "KILLPRO" qua server thông qua hàm sendSignal("KILLPRO", cSocket) và hiển thị form của class KILL.

+Hàm *jButton3ActionPerformed()* được thực hiện khi nhấn nút DELETE, lúc này xóa hết các dòng của Table ở cửa sổ Process

+ Hàm *jButton4ActionPerformed()* được thực hiện khi nhấn nút START, lúc này sẽ gửi tín hiệu "STARTAPP" qua server thông qua hàm sendSignal("STARTAPP", cSocket) và hiển thị form của class START.

7. START (*START.java*)

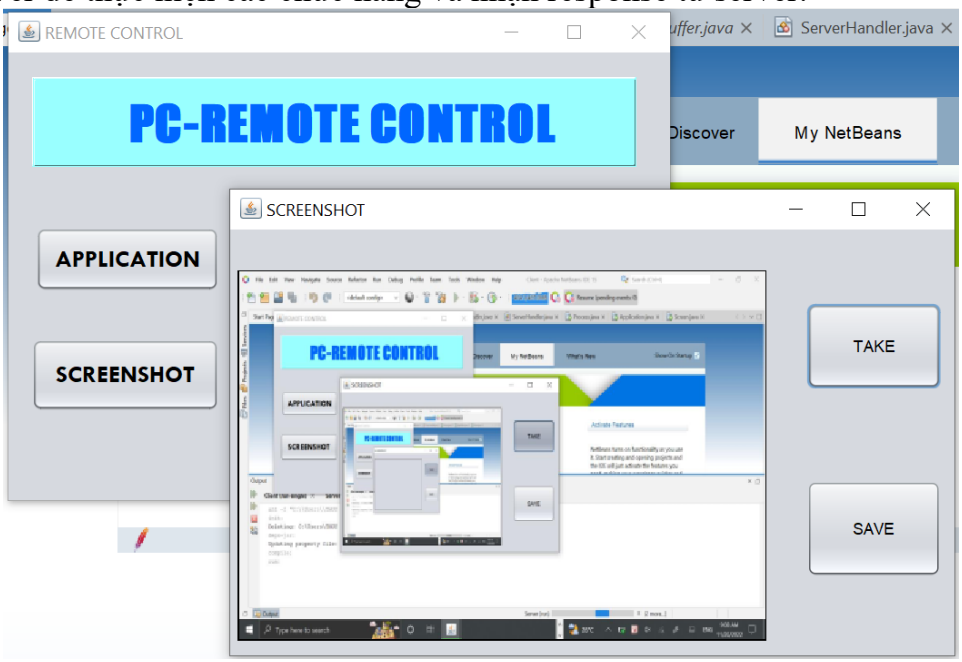
Chức năng: Hiển thị form để xử lí, gửi tên app cần bật đến server và nhận tín hiệu thành công hay không từ server.

8. Screen (*Screen.java*)

Chức năng

Hiển thị form gồm các nút bấm để xử lí, gửi tín hiệu đến server để thực hiện các chức năng liên quan đến ScreenShot: Chụp ảnh màn hình của máy Server (TAKE) và lưu

ảnh chụp màn hình vào vị trí tùy chọn trong máy Client (SAVE). Gửi request qua server để thực hiện các chức năng và nhận response từ server.



Các hàm, thủ tục

+ Khai báo các biến cSocket, dOut, dIn, img và tạo Constructor Screen với tham chiếu cSocket:

```
25      Socket cSocket;
26      DataOutputStream dOut;
27      DataInputStream dIn;
28      BufferedImage img;
29
30      public Screen(Socket cSocket) throws IOException {
31          initComponents();
32          this.cSocket = cSocket;
33          this.dOut = new DataOutputStream( out: cSocket.getOutputStream());
34          this.dIn = new DataInputStream( in:cSocket.getInputStream());
35      }
```

+ Hàm sendSignal(String s, Socket sc) với tham chiếu s, sc dùng để ghi dữ liệu để gửi request cho server thông qua biến dOut.

```
37      public void sendSignal(String s, Socket sc) throws IOException {
38          dOut.writeUTF( str:s);
39          dOut.flush();
40      }
```

+ Hàm jButton1ActionPerformed() được thực hiện khi nhấn nút TAKE, lúc này sẽ gửi tín hiệu "SCREENSHOT" qua server thông qua hàm sendSignal("SCREENSHOT", cSocket), server nhận request "SCREENSHOT" từ client thông qua và trả về response thông qua DataOutputStream. Khi nhận request từ client, server sẽ tiến hành chụp màn

hình và lưu vào bộ nhớ tạm dưới dạng mảng. Client sau đó tiến thành tạo một biến có kích thước phù hợp và đọc mảng được lưu tạm này vào biến với kích thước vừa đủ. Từ biến đó chuyển thành hình ảnh dạng icon và hiển thị lên cửa sổ của chức năng Screenshot để người dùng ở client có thể xem ảnh vừa chụp trước khi lưu. Do đó người dùng có thể chụp rất nhiều ảnh chụp màn hình cho đến khi quyết định lưu.

```
127     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
128         try {
129             sendSignal( s:"SCREENSHOT", sc:cSocket);
130             DataInputStream in = new DataInputStream( in:cSocket.getInputStream());
131             DataOutputStream out = new DataOutputStream( out:cSocket.getOutputStream());
132 
133             int size = in.readInt();
134             byte[] imageAr = new byte[size];
135             in.readFully( b:imageAr);
136 
137             img = ImageIO.read(new ByteArrayInputStream( buf:imageAr));
138             ImageIcon imgicon = new ImageIcon( image:img);
139             Image imfit = imgicon.getImage();
140             Image imgfit = imfit.getScaledInstance( width:jLabel1.getWidth(),  height:jLabel1.getHeight(),  hints:Image.SCALE_SMOOTH);
141             jLabel1.setIcon(new ImageIcon( image:imgfit));
142         } catch (IOException ex) {
143             Logger.getLogger( name:Screen.class.getName()).log( level:Level.SEVERE,  msg:null,  thrown:ex);
144         }
145     }
```

+ Hàm jButton2ActionPerformed() được thực hiện khi nhấn nút SAVE, lúc này trong Client sử dụng chức năng File Choose để hiển thị hộp chọn địa chỉ lưu. Sau đó người dùng sẽ chọn vị trí muốn lưu ảnh trong hộp và nhập tên của ảnh theo ý muốn, kèm đuôi có định dạng phù hợp để lưu ảnh thành công.

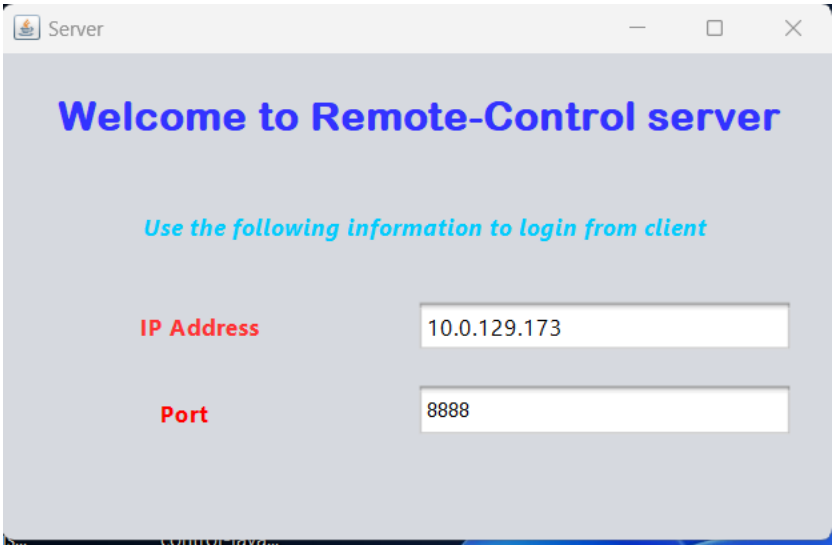
```
109     private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
110         // TODO add your handling code here:
111         JFrame parentFrame = new JFrame();
112         JFileChooser directoryChooser = new JFileChooser();
113         directoryChooser.setDialogTitle( dialogTitle:"Specify a file to save");
114         Integer returnVal = directoryChooser.showSaveDialog( parent:parentFrame);
115 
116         if (returnVal == JFileChooser.APPROVE_OPTION)
117         {
118             File fileToSave = directoryChooser.getSelectedFile();
119             try {
120                 ImageIO.write( im:img, formatName:"jpg", output:fileToSave);
121             } catch (IOException ex) {
122                 Logger.getLogger( name:Screen.class.getName()).log( level:Level.SEVERE,  msg:null,  thrown:ex);
123             }
124         }
125     }
```

II. Project: Server

Package: server

1. ServerFrame (*ServerFrame.java*)

Chức năng: Hiển thị form để hiển thị địa chỉ IP và port 8888 (port cố định) của máy server. Gọi class ServerHandler.



2. ServerHandler (*ServerHandler.java*)

Chức năng:

a) Khởi tạo ServerSocket với port 8888 (port cố định)
b) Accepct kết nối từ client để thực hiện nhận request từ client và trả về reponse.
c) Xử lí, thực hiện các lệnh liên quan đến chức năng khi nhận được request thực hiện chức năng đó từ client.
d) Trả về kết quả thực hiện chức năng về cho client.

Các lệnh thực hiện các chức năng:

+ Khai báo biến clients để đánh dấu để kết thúc vòng lặp bên dưới, nghĩa là khi nhận tín hiệu "EXIT" từ client, biến clients sẽ tăng lên 1 để thoát vòng lặp. Biến serverC được khởi tạo, biến client dùng để chấp nhận kết nối từ phía client nếu đúng IP, Port.

+ Trong vòng while (clients == 0) sẽ đọc các request từ client (khi phía client bấm vào các nút ở form của ClientHandler) gửi qua server và lưu vào biến *re*.

+ Phần lệnh if (re.equals("APP")), nếu nhận được request "APP" từ cửa sổ form ClientHandler ở phía client thì sẽ xử lý những chức năng: list running apps, kill running apps bằng ID, start app ở bên máy server khi nhận được những request ở phía client ("LISTAPP", "KILLAPP", "STARTAPP"). Bên trong lệnh if có 1 vòng lặp while (checkApp) dùng để lặp cho đến khi nhận được request "EXITAPP" từ client, tức là bên client đóng cửa sổ form Application. Đọc các request từ cửa sổ form Application của phía client và lưu vào biến *aRe*.

- Phần lệnh bên trong if (aRe.equals("LISTAPP")) sẽ thực hiện chức năng chạy process theo lệnh yêu cầu trong powershell bằng cách dùng biến Process, ProcessBuilder; đếm số dòng và đọc từng dòng kết quả trả về khi

chạy lệnh trong powershell; gửi số lượng dòng, nội dung từng dòng về client thông qua dOut.

- Phần lệnh bên trong if (aRe.equals("KILLAPP")) sẽ thực hiện chức năng: nhận ID của app cần KILL bên client, xử lí chạy process theo lệnh cmd.exe để kill app bằng ID *"taskkill /PID {ID} /F"* bằng cách dùng các biến Process, ProcessBuilder và trả kết quả thực hiện bằng các byte 0, 1 (0: không thành công; 1: thành công).
- Phần lệnh bên trong if (aRe.equals("STARTAPP")) sẽ thực hiện chức năng: nhận tên app cần bật từ client và sau đó xử lí chạy process theo lệnh cmd.exe để bật app bằng tên: "start {name}.exe" bằng cách dùng các biến Process, ProcessBuilder và trả kết quả thực hiện bằng các byte 0, 1 (0: không thành công; 1: thành công).

+ Phần lệnh if (re.equals("PRO")), nếu nhận được request "PRO" từ cửa sổ form ClientHandler ở phía client thì sẽ xử lý những chức năng: list running process, kill running process bằng PID, start process ở bên máy server khi nhận được những request ở phía client ("LISTPRO", "KILLPRO", "STARTPRO"). Bên trong lệnh if có 1 vòng lặp while (checkPro) dùng để lặp cho đến khi nhận được request "EXITAPP" từ client, tức là bên client đóng cửa sổ form Process. Đọc các request từ cửa sổ form Process của phía client và lưu vào biến *aRe*.

- Phần lệnh bên trong if (aRe.equals("LISTPRO")) sẽ thực hiện chức năng chạy process theo lệnh yêu cầu trong cmd bằng cách dùng biến Process, ProcessBuilder; đếm số dòng và đọc từng dòng kết quả trả về khi chạy lệnh trong cmd; gửi số lượng dòng, nội dung từng dòng về client thông qua dOut.
- Phần lệnh bên trong if (aRe.equals("KILLPRO")) sẽ thực hiện chức năng: nhận ID của app cần KILL bên client, xử lí chạy process theo lệnh cmd.exe để kill process bằng ID *"taskkill /F /PID +{ID};* bằng cách dùng các biến Process, ProcessBuilder và trả kết quả thực hiện bằng các byte 0, 1 (0: không thành công; 1: thành công).
- Phần lệnh bên trong if (aRe.equals("STARTAPP")) sẽ thực hiện chức năng: nhận tên process cần bật từ client và sau đó xử lí chạy process theo lệnh cmd.exe để bật process bằng tên: "start {name}.exe" bằng cách dùng các biến Process, ProcessBuilder và trả kết quả thực hiện bằng các byte 0, 1 (0: không thành công; 1: thành công)

+ Phần lệnh if (re.equals("SCREENSHOT")), nếu nhận được request "SCREENSHOT" từ cửa sổ form ClientHandler ở phía client thì sẽ xử lý chức năng: Chụp màn hình và lưu ảnh vào Output Stream dưới dạng Buffer. Bên trong lệnh if có 1 vòng lặp while (checkApp) dùng để lặp cho đến khi nhận được request "EXITSCREEN" từ client, tức là bên client đóng cửa sổ form Screenshot. Đọc các request từ cửa sổ form Screenshot của phía client và lưu vào biến *aRe*.

+ Phần lệnh if (re.equals("SHUTDOWN")):

- **Hàm runtime.getRuntime():**

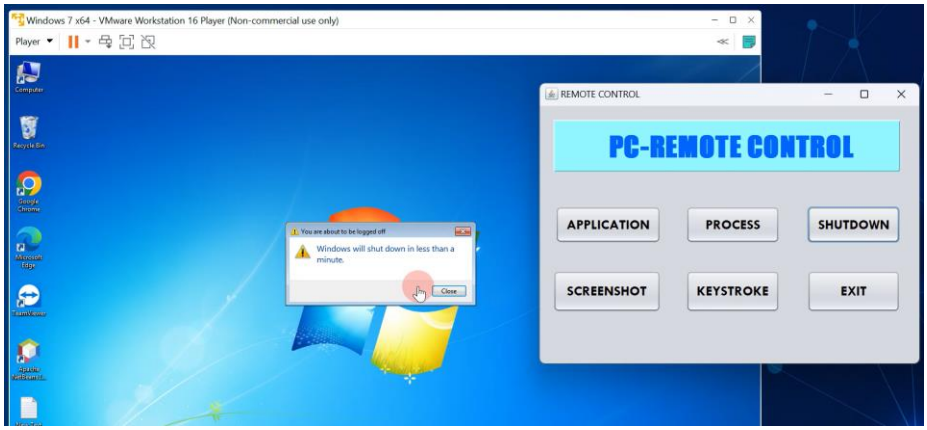
Chức năng: Trả về đối tượng thời gian chạy được liên kết với phần mềm Java hiện tại.

Cách hoạt động: Lấy thời gian chạy hiện tại được liên kết với tiến trình, chương trình đang chạy.

- **Hàm runtime.exec(command):**

  Chức năng: Thực thi lệnh chuỗi đã chỉ định trong một quy trình riêng biệt. Phương thức này trả về một đối tượng tiến trình mới để quản lý tiến trình con.

  Cách hoạt động: Cụ thể trong bài làm, hàm sẽ là **runtime.exec("shutdown -s -t 5")**, và liên kết thực thi trong Command Prompt, sử dụng Shutdown command để điều khiển tắt máy server. Trong Shutdown command có hỗ trợ các lệnh tắt máy "**-s**" và hẹn giờ thực thi lệnh "**-t xxx**" (xxx: từ 0 đến 315360000 giây). Bài làm cài đặt tắt máy sau 5 giây để server có thể hiện thông báo tắt máy vừa đủ thời gian đọc. Sau khi thực hiện, màn hình sẽ thông báo máy của bạn sẽ tắt trong vòng vài giây/ít hơn 1 phút "Windows will shutdown in less than a minute." (ảnh minh họa bên dưới)



- *Lỗi* IOException*:* Nếu xảy ra Input/Output Error, chương trình sẽ in ra Exception cụ thể.