جامعة الأخوين

ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵓⵅⴰⵡⴰⵢⵏ

AL AKHAWAYN UNIVERSITY

RSC (Remote System Controller)

Anass Zouine

CSC 3374

Professor Iraqi Houssaini Omar

4th March 2021

Table of Contents:

1. Purpose of the Application

This client-server application allows taking control of a remote system. It allows.

- Getting the name and version of the operating system of the remote machine
- Taking screenshots of the remote system
- Rebooting the remote system

2. End-User Manual

2.1. Running the server:

The directory of the applications is called **RSCapp**. It contains the server program compiled named **RSCserver**. Open a terminal session and run this file.

2.2. Running the client:

The directory of the applications is called RSCapp. It contains the client program compiled named **RSCclient**. When you run this file, you should pass an argument to it that will specify the operation that you choose for the client program to do. The following are the possible arguments that you can pass at the command line when you run RSCclient:

a. get: to get name and version of OS of the remote system.
b. reboot: to reboot the remote system.
c. screenshot: to take screenshots of the remote systems screen.
d. help: to print the available arguments that the RSCclient accepts and the operations associated with each one of them.

3. Protocol

3.1. Protocol

1. The client opens a connection with the server and informs the server of which operation it wants to request:
2. If the client wants to get the name and version of the OS of the remote machine, the header will be as the following: **Get[line feed]**
   2.1. Upon receiving this header, the server shall retrieve the OS name and version.
   2.2. If the server retrieves the OS properties successfully, it shall respond with a header as the following: **[OS Name][semicolon][OS Version][line feed]**

2.3. If the server is unable to retrieve the OS properties, it shall respond with a header as the following: **Unavailable[line feed]**

3. If the client wants to take a screenshot of the remote system:
   3.1. The header will be as the following: **Screenshot[line feed]**
   3.2. Upon receiving this header, the server shall take a screenshot of the remote system
   3.3. If the screenshot is not taken successfully, the server shall send a header to the client as the following: **No[line feed]**
   3.4. If the server took the screenshot successfully, it should send a header to the client as the following: **Yes[line feed]**
   3.5. After sending the header of successful operation, the server should send the actual bytes that make the screenshot taken
4. If the client wants to reboot the remote system:
   4.1. The header will be as the following: **Reboot[line feed]**
   4.2. Upon sending the header, the client shall keep checking if the remote host is reachable.
   4.3. Upon receiving the header, the server shall reboot the remote system,
   4.4. If the client detects that the remote system is no more reachable, then the server rebooted it successfully.
   4.5. If the server is unable to reboot the system, the client will keep detecting that the remote system is reachable, and it will receive a header from the server as the following: **Unable[line feed]**

***Clarification about 4.4**: there is no way for the server to communicate through a header that the reboot was done successfully, because it will stop executing if the remote system shutdown. For this reason, the server needs the client to check if the remote host is reachable, so that if it is not, the client understands that the remote system was rebooted successfully.

4. Code Structure

4.1. RSCclient structure

The client program of the application starts by opening the connection actively by creating an instance of socket class passing to it the IP address of the machine where the server is hosted and the port to which it is listening.

After, the program reads the argument passed through the command line by the end user (vertical communication between end-user and client). Based on the argument passed, the client communicates (horizontal communication) with the server to perform the appropriate operation:

a. Get name and version of the operating system of the remote system

The appropriate header is created in string format and is sent to the server. Then, the client reads the response header sent by the server. If the header is equal to “Unavailable”, the client understands that the properties of the OS cannot be retrieved, and an error message is printed to the end user. Otherwise, the header will be containing the OS properties. Then, the client extracts the name and the version of the operating system and prints them to the end-user.

b. Take a screenshot of the remote system

The appropriate header is created in string format and is sent to the server. Then, the client reads a header sent by the server. If the header is equal to “No” then the client prints a message to the end user that the screenshot could not be taken. If it is equal to “Yes”, then it proceeds to receive the image and save it. It gets the current date and converts it to string format. A new file is created under the directory RSC that is inside the directory of the application. The file is named the current date to guarantee that screenshot files will always have a unique name. Finally, the client reads the image sent by the server into the PNG file and saves it.

c. Reboot the remote system

The appropriate header is created in string format and is sent to the server. The client starts a loop and keeps checking if the remote system is reachable (not yet rebooted) by sending ping requests. As long as it is reachable, the client keeps checking if a header is sent by the server. If the header “Unable\n” is received, then the client breaks from the loop and announces to the end-user that the remote system was not rebooted successfully. If the client detects that the remote system is not reachable, then it ends the loop and announces to the end user that the remote system is rebooted successfully.

4.2. RSCserver structure

The server program of the application starts by creating an instance of ServerSocket class. Since the server is supposed to be running forever, the code is put inside an infinite loop. Inside the loop, the server opens the connection passively through a blocking call waiting for connection request from the client. When a connection is actively opened by a client, a message is printed to the output specifying the IP address of the machine of client and the port number to which it is listening. The server reads the header sent by the client and extracts the command that the client requested. Based on the command extracted from the header, the server performs the appropriate operation:

a. Get:

The server gets the name and the version of the operating system. If an exception is raised, the server sends a header to the client to inform that properties of the OS cannot be retrieved. If no exception is raised, the name and version are concatenated and grouped in a header properly as the protocol states and sent back to the client.

b. Screenshot:

The server takes a screenshot image and stores it as an instance of BufferedImage. If an exception is raised, the server sends a header to inform the client that the screenshot could not be taken. If no exception is raised, the server

sends a header to confirm that screenshot is taken. Then, the buffered image is sent to the client through the output stream.

c. **Reboot:**

The server calls a user defined function which is reboot. The function gets the name of the operating system of the remote machine on which the server is running. Based on the name of the OS, a command is created as a string and then executed. If an exception is raised, then the server sends a header to the client informing that it could not reboot the remote system. Otherwise, the remote system on which the server is hosted will be rebooted, and no header will be sent to the client (because the system will be rebooting).