Socket Programming

Reflective Report

Joe Morais (K00254840)

Software Development Year 2

Internetworking

Group Project

Contents

	Page
Introduction	3
Research	3
Running the Utilities	3
Networking Aspects	4
Coding & Implementation	4
Future Implementations	5
Final Notes	6

Introduction

The purpose of this report is to outline and describe the steps taken to complete the Socket Programming assignment for the Internetworking module.

The project requirements were to design and implement two client-server network utilities using Python while working in a group.

The final version of our application included four Python files. Two for each of the utility tasks.

Research

To complete this assignment, our group researched the use of the Python socket library and tested several examples found online. We eventually created a simple version of the application that established the connection between the client and the server.

We then added more functionality according to the project requirements and finally added the functionality of allowing the server to continuously listen until the client closes the connection.

One of the problems we encountered for the first task, was specifying the message size, since this variable must be provided to the ***recv()*** function. We considered creating our own protocol where we could calculate the length of the string and pass that to the server along with the message. The server could then split the message and use the provided length for the **recv()** function.

After some further research, we decided that 1024 bytes would be more than enough for the purpose of the first Client/Server application. The second application uses 16 bytes.

Running the Utilities

To run each utility, the user must first run the server and then the client in a separate terminal. Running the client first will result in an error due to the client expecting to be able to connect to the server when it runs.

The server then waits for the client to connect and send a message. On the client side, the user is prompted with a message. The client establishes a connection with the server and sends a message. The server replies with a result based on the user's input.

Networking Aspects

While creating the scripts, we were required to establish a connection between the server and the client on the localhost. We used knowledge gained from the Internetworking module as well as previous modules related to networking. The steps taken for both client and server were:

Server Side

- Creating a TCP/IP connection.
- Binding the socket to a port.
- Continuously listening for incoming connections and waiting for a client to connect.
- Receiving data from the client.
- Sending a reply to the client.
- Closing the connection.

Client Side

- Creating TCP/IP connection.
- Connecting to the port where the server is listening.
- Sending data to the server.
- Waiting for a response from the server.
- Closing the connection.

Coding & Implementation

We worked on the project one step at a time and ended up with about five different version of the scripts. The final version (1.5) was the version we uploaded for grading.

The main steps taken to complete the final version of the app involved:

- Establishing a connection between the server and client.
- Sending messages from the client to the server.
- Sending replies from the server to the client.
- Creating functions to check the user's input and testing them without the server/client scripts.
- Implementing the functions in the server scripts.
- Creating a simple, console-based interface to allow the user to interact with the client.
- Creating a loop that would allow the server to receive multiple requests from the client before closing the connection.

For the first client/server, the user can enter a string of characters on the client side. The client then establishes the connection and sends the string to the sever. The server receives the message and checks if the characters are all alphabetic (spaces are allowed). It then sends a reply to the client with the result. If the string passes the test, the server sends the string back to the client. If the string fails the test, the server sends a "False" response.

For the second client/server, the user is prompted to enter an integer on the client side. The client then establishes the connection and sends the integer to the server. The server receives the integer and checks if the integer is a power of 2. If the integer passes the test, the server sends a "True" response to the client. If the integer fails the test, the server sends a "False" response. The user can provide four integers, one at a time. Typing "exit" into the console will terminate the application.

Note: Input validation was not implemented on the server side to check if the user entered an integer or not. It attempts to convert the input to an integer. Entering any other data type will cause the application to crash.

Future Implementations

Our scripts were extensively tested and work as expected. However, there are several things that could be implemented in future versions to improve the functionality and reliability of the utilities. Some of which include:

Client to Server Error Handling

The server must always be running when the client attempts to connect. Implementing error handling on the client side would fix this issue.

Input Validation

Checking whether the user has entered an integer or not would avoid crashing the application.

Specifying Bytes

Calculating the correct amount of bytes required for the string, and somehow passing that to the servers; so it can use it in the ***recv()*** function, would avoid issues where the user enters more bytes than the server is expecting.

Final Notes

This project helped me in several ways, from both a network and software development perspective. The skills gained can be used across several real-world applications. Some of the most valuable skills I gained from completing this project were:

- Working in group to complete a project's specifications.
- Using Python's socket library.
- Establishing a connection between a server and client over the local network using Python.
- Handling and processing data in Python between two separate applications over the network.