Principles of Computer Communications
Project 2 Report

Emre Çamlıca - 150210071

1 Introduction

In this project, I designed client and server applications with socket programming. In the program, the client first sends a text file to the server. The server stores this file to be sent back later. Then, the client makes another connection to receive the file back. If the contents of the sent and the received files are the same, the program generates a successful message, otherwise, it generates an error message.

2 Client Code

Figure 1 shows the client code. In the code, I first imported the required libraries, I then used and edited the sample code provided in the textbook for socket programming to connect the client to the server. The client first asks the user which option to perform between sending a file and comparing the file contents. After that, it sends the option to the server. Here, I added a sleep statement to prevent the server from conflicting the option message with the contents of the file. Then, the client executes the related part of the code depending on the option. If option 1 is selected, the client reads the contents of the client file and sends it to the server, encoding it in binary form. If option 2 is selected, the client receives a file as an encoded binary message from the server, converts it into a string, and writes it back to another file. Finally, using the "filecmp" function, the client compares the contents of this file to the client file. If the contents are the same, a successful message is printed. If not, a failure message is printed.

```python
from socket import *
import filecmp
import time
serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
option=input("Choose 1 to send a file to server, 2 to check your file:")
clientSocket.connect((serverName, serverPort))
clientSocket.send(option.encode())
time.sleep(1)

if int(option) == 1:
    with open("client.txt", "r") as file:
        contents = str(file.read(1024))
    file.close()
    clientSocket.send(contents.encode())
    clientSocket.close()

elif int(option) == 2:
    filecheck=clientSocket.recv(1024).decode()
    with open("filecheck.txt", "w") as file:
        file.write(filecheck)
    file.close()

    file1 = "client.txt"
    file2 = "filecheck.txt"

    if filecmp.cmp(file1, file2, shallow=False):
        print("Transmission successful.")
    else:
        print("Transmission failed.")
    clientSocket.close()
```

Figure 1: Client Code

3 Server Code

The server code is initialized similarly. The "listen(1)" statement allows for only one client to be connected to the server at a time. The "while True" statement is used so that the server keeps accepting clients one after another. The server first receives the option from the client. If the first option is sent, the server receives the contents of the client file in binary, decodes it, converts it into string, and rewrites that to the server file. If the second option is sent, the server stores the contents of the server file, encodes it, and sends it to the client program. An error message is generated if the value of the option is not meaningful. The server closes the connection socket after each interaction with the client is completed.

```python
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)

while True:
    connectionSocket, addr = serverSocket.accept()
    print("Choose 1 to send a file to server, 2 to check your file:")
    option=int(connectionSocket.recv(1024).decode())

    if option == 1:
        print("option 1 is chosen.")
        contents = str(connectionSocket.recv(1024).decode())
        with open("server.txt", "w") as file:
            file.write(contents)
        file.close()
        connectionSocket.close()

    elif option == 2:
        print("option 2 is chosen.")
        with open("server.txt", "r") as file:
            contents = str(file.read(1024))
        file.close()
        connectionSocket.send(contents.encode())
        connectionSocket.close()

    else:
        print("error")
        connectionSocket.close()
```

Figure 2: Server Code

4 Conclusion

The client and server application works as expected. If the client did not change the file contents after sending it to the server, the program displays an affirmative message, otherwise, it displays a negative message, as can be observed from the video recordings. This project helped me take a glance at how things work in practice in the application layer and I benefited from the textbook and the open resources on the internet in doing this.