Distributed Systems: Homework 1 Report

cmtidmar, cmtidmar@iu.edu

February 2022

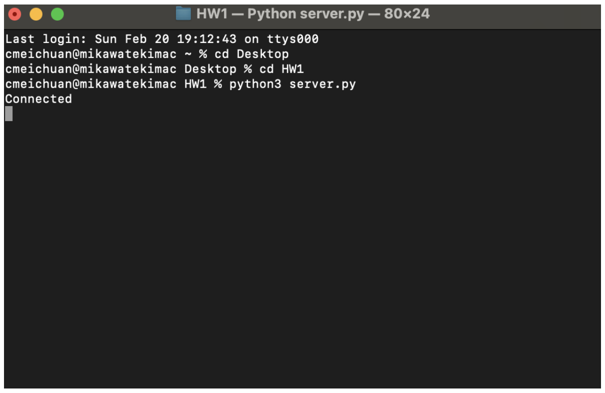
1 Introduction

Assignment 1 for CSCI-P434 Spring 2022 at Indiana University.

2 Design Details with Examples

2.1 Server

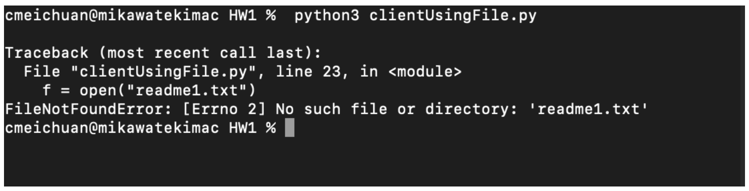
The first thing to do on the server-side was to initialize a port that the socket would be able to use. After initializing the the port, a socket as immediately created then binded to a server host as well as the server port. The server host was not given a value as it would be up the client where they want to host this program. The newly created, or welcoming socket, then listens to TCP requests where the parameter of listen() is 1, or true. If the socket is connected, a print statement will appear in the user's terminal displaying a message to notify the user that the socket was successfully connected.



```
Last login: Sun Feb 20 19:12:43 on ttys000
cmeichuan@mikawatekimac ~ % cd Desktop
cmeichuan@mikawatekimac Desktop % cd HW1
cmeichuan@mikawatekimac HW1 % python3 server.py
Connected
```

Figure 1: Connected To Server

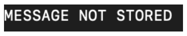
While the socket is connected, a temporary/connection socket is created and accepts the incoming requests from a client. The program will then notify the user that the creation of a new socket was successful. After the socket accepts the client's request, the connection socket will receive the message from the client and will be stored in the 'dataFromClient' variable. The message from the client is encoded during this communication, so a variable, "message", was created to decode then temporarily hold the message the client had sent. If there is no message or the text file that the user inputs does not exist, the user will be notified that the message was not stored.



```
cmeichuan@mikawatekimac HW1 %  python3 clientUsingFile.py

Traceback (most recent call last):
  File "clientUsingFile.py", line 23, in <module>
    f = open("readme1.txt")
FileNotFoundError: [Errno 2] No such file or directory: 'readme1.txt'
cmeichuan@mikawatekimac HW1 %
```

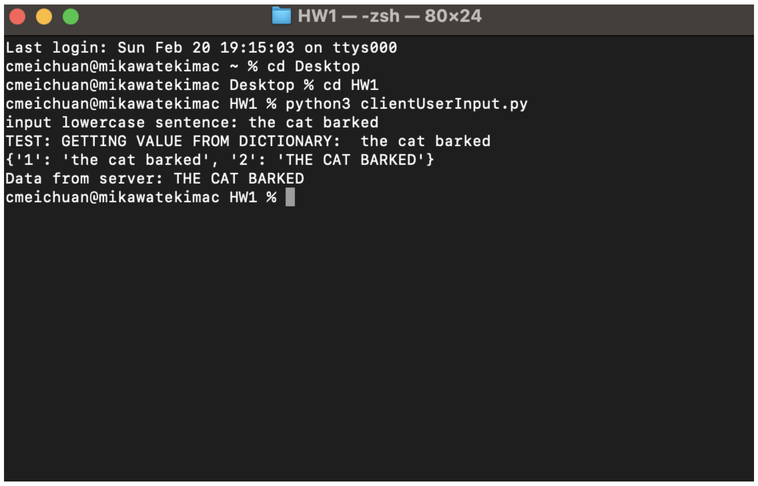
Figure 2: Client Gives Server Nonexistent File



```
MESSAGE NOT STORED
```

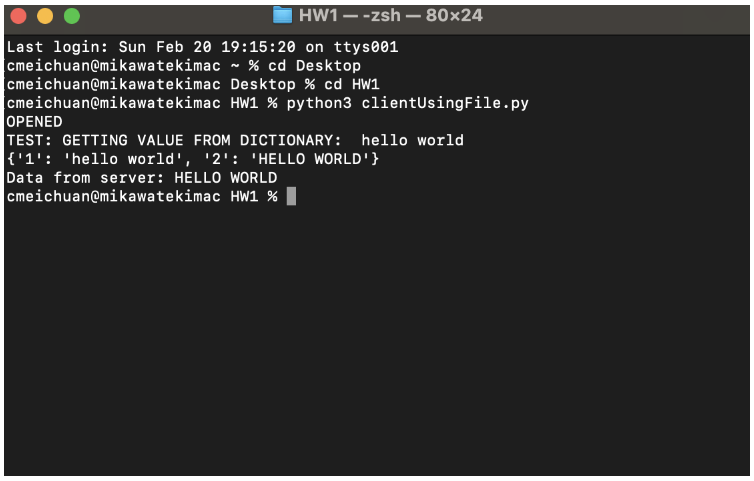
Figure 3: Server's Response to Figure 2

However, if the client sends a message, then the user will be notified that the message was able to decode and stored. After storing the message from the client, the server will take the data from the client and translate the message into all uppercase characters. After translating the message, the server will then send the new message back to the client and the connection socket will close.



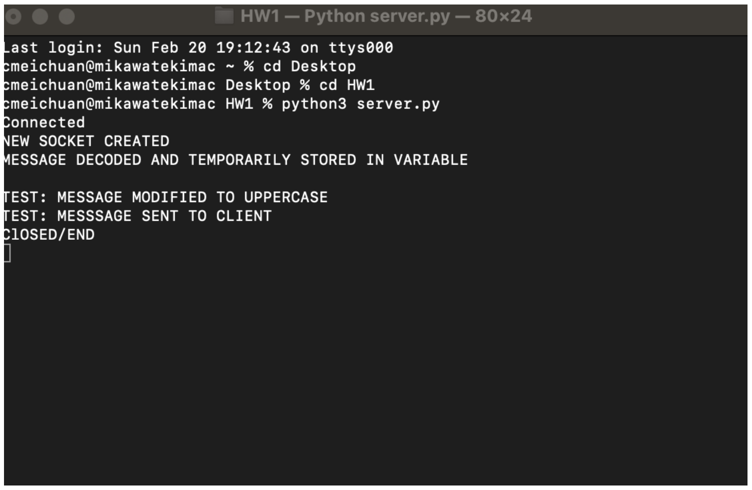
```
Last login: Sun Feb 20 19:15:03 on ttys000
cmeichuan@mikawatekimac ~ % cd Desktop
cmeichuan@mikawatekimac Desktop % cd HW1
cmeichuan@mikawatekimac HW1 % python3 clientUserInput.py
input lowercase sentence: the cat barked
TEST: GETTING VALUE FROM DICTIONARY:  the cat barked
{'1': 'the cat barked', '2': 'THE CAT BARKED'}
Data from server: THE CAT BARKED
cmeichuan@mikawatekimac HW1 %
```

Figure 4: Client (User Input) Sends To and Receives From Server



```
Last login: Sun Feb 20 19:15:20 on ttys001
cmeichuan@mikawatekimac ~ % cd Desktop
cmeichuan@mikawatekimac Desktop % cd HW1
cmeichuan@mikawatekimac HW1 % python3 clientUsingFile.py
OPENED
TEST: GETTING VALUE FROM DICTIONARY:  hello world
{'1': 'hello world', '2': 'HELLO WORLD'}
Data from server: HELLO WORLD
cmeichuan@mikawatekimac HW1 %
```

Figure 5: Client (File Input) Sends To and Receives From Server



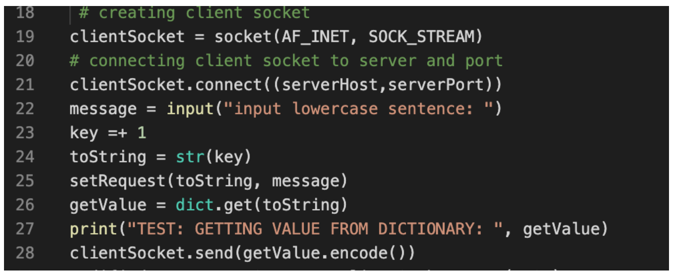
```
ast login: Sun Feb 20 19:12:43 on ttys000
cmeichuan@mikawatekimac ~ % cd Desktop
cmeichuan@mikawatekimac Desktop % cd HW1
cmeichuan@mikawatekimac HW1 % python3 server.py
onnected
EW SOCKET CREATED
ESSAGE DECODED AND TEMPORARILY STORED IN VARIABLE

EST: MESSAGE MODIFIED TO UPPERCASE
EST: MESSSAGE SENT TO CLIENT
lOSED/END
```

Figure 6: Server Accepts Message From Client

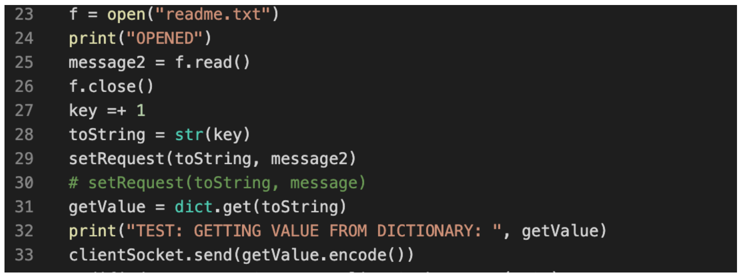
2.2 Client

Two separate clients were created: clientUserInput and clientUsingFile in order to demonstrate the usage of the multiple clients on one server as well as two different ways the client can do to input a value that will eventually be sent to the server. The two clients are almost identical with only a few minor changes. Both clients initialize the server host and server port. They both have two functions getRequest and setRequest. However, getRequest was not used. Both clients also initialize their own client sockets where they then will connect to the server host and port. Next, in the clientUserInput file, the user will be prompted to input a message in all lowercase. The message will then be sent to the server and the server will then send back the same message in all uppercase (see Figure 4 for example). The clientUsingFile file will not prompt the user to input message, but rather it will open a file and read the contents from it. The contents will then be sent to the server as a message and then will be sent back with the same result as the clientUserInput.



```
# creating client socket
clientSocket = socket(AF_INET, SOCK_STREAM)
# connecting client socket to server and port
clientSocket.connect((serverHost,serverPort))
message = input("input lowercase sentence: ")
key =+ 1
toString = str(key)
setRequest(toString, message)
getValue = dict.get(toString)
print("TEST: GETTING VALUE FROM DICTIONARY: ", getValue)
clientSocket.send(getValue.encode())
```

Figure 7: Code Snippet of User Input



```
f = open("readme.txt")
print("OPENED")
message2 = f.read()
f.close()
key =+ 1
toString = str(key)
setRequest(toString, message2)
# setRequest(toString, message)
getValue = dict.get(toString)
print("TEST: GETTING VALUE FROM DICTIONARY: ", getValue)
clientSocket.send(getValue.encode())
```

Figure 8: Code Snippet of Reading From File

The final result from both clients will be a dictionary that holds both the original message that was sent to the server and the new message the server sends to the client (see Figures 4 and 5).

3 Server Limitations

This server is very basic and with many limitations. There are no key or value size limits as the user is prompted to input a rather short message/sentence. The key will always be an integer converted to string. It also does not handle the situation if a user were to input uppercase letters. This server could be modified to the where if the user does input all uppercase then the server will send the client all lowercase. The server could also be modified to handle the situation if the user were to input some upper- and some lower- case characters. The server could then randomize its message back to the client either send all uppercase or all lowercase. The server does not handle the cases when a client might send numeric values or characters that cannot be upper- or lower-case. There are also no error exceptions that are handled throughout the program. Since this server, client interaction is very basic at its core, both the client and server could be modified in the future to handle more specific, or detailed, instances.