Dillon Tate and Andrew Donovan

CPE 586 Lab 2

Webserver Executions for part 1:

Text

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

For the first webserver, we simply filled in the blanks with the appropriate sections of code. We used the port 32008 for our main server, with IP address 127.0.0.1. The call for this is shown as above, “http://localhost:32008/helloworld.html”. For another file that doesn’t exist, “fake.html”, a 404 error was returned. The executions of this part are as above.

Proxy server executions for part 2:

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application, website

Description automatically generated

For the second part of this project, we copied the initial server and added a cache list. This would take the request and forward it to the initial server by appending http:/ to the extension. So, the call to the proxy was “http://localhost:32007/localhost:32008/helloworld.html” and the proxy would request “http://localhost:32008/helloworld.html” and return the data from the server to the initial client. To make this call we used the requests library to call it as an HTTP GET request and return the file data as binary. The file was treated as binary so other file types (besides html) could also be retrieved. The source code for both problems is below, and sample executions are shown above.

Source code for part 1:

# Import socket module

from socket import \*

# Create a TCP server socket

#(AF\_INET is used for IPv4 protocols)

#(SOCK\_STREAM is used for TCP)

serverSocket = socket(AF\_INET, SOCK\_STREAM)

# Fill in start

# Defining serverSocket HOST and PORT, and listening

HOST = '127.0.0.1'

PORT = 32008

serverSocket.bind((HOST, PORT))

serverSocket.listen()

# Fill in end

# Server should be up and running and listening to the incoming connections

while True:

print('Ready to serve...')

# Set up a new connection from the client

connectionSocket, addr = serverSocket.accept() #Fill in start #Fill in end

# If an exception occurs during the execution of try clause

# the rest of the clause is skipped

# If the exception type matches the word after except

# the except clause is executed

try:

# Receives the request message from the client

message = connectionSocket.recv(1024) #Fill in start #Fill in end

# Extract the path of the requested object from the message

# The path is the second part of HTTP header, identified by [1]

filename = message.split()[1]

# Because the extracted path of the HTTP request includes

# a character '\', we read the path from the second character

f = open(filename[1:])

# Store the entire contenet of the requested file in a temporary buffer

outputdata = f.read() #Fill in start #Fill in end

# Send the HTTP response header line to the connection socket

# Fill in start

# Connection is successful if it gets here

connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n".encode())

# Fill in end

# Send the content of the requested file to the connection socket

for i in range(0, len(outputdata)):

connectionSocket.send(outputdata[i].encode())

connectionSocket.send("\r\n".encode())

# Close the client connection socket

connectionSocket.close()

except IOError:

# Send HTTP response message for file not found

# Fill in start

connectionSocket.send("HTTP/1.1 404 Not found\r\n\r\n".encode())

# Fill in end

# Close the client connection socket

# Fill in start

connectionSocket.close()

# Fill in end

serverSocket.close()

Source code for part 2:

Web Server:

# Import socket module

from socket import \*

# Create a TCP server socket

# (AF\_INET is used for IPv4 protocols)

# (SOCK\_STREAM is used for TCP)

serverSocket = socket(AF\_INET, SOCK\_STREAM)

# Fill in start

# Defining serverSocket HOST and PORT, and listening

HOST = '127.0.0.1'

PORT = 32008

serverSocket.bind((HOST, PORT))

serverSocket.listen()

# Fill in end

# Server should be up and running and listening to the incoming connections

while True:

print('Ready to serve...')

# Set up a new connection from the client

connectionSocket, addr = serverSocket.accept() # Fill in start #Fill in end

# If an exception occurs during the execution of try clause

# the rest of the clause is skipped

# If the exception type matches the word after except

# the except clause is executed

try:

# Receives the request message from the client

message = connectionSocket.recv(1024) # Fill in start #Fill in end

# Extract the path of the requested object from the message

# The path is the second part of HTTP header, identified by [1]

filename = message.split()[1]

# Because the extracted path of the HTTP request includes

# a character '\', we read the path from the second character

f = open(filename[1:], "rb")

# Store the entire content of the requested file in a temporary buffer

outputdata = f.read()

# close the file

f.close()

# Send the HTTP response header line to the connection socket

# Connection is successful if it gets here

connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n".encode())

# Send the content of the requested file to the connection socket

connectionSocket.sendall(outputdata)

# for i in range(0, len(outputdata)):

# connectionSocket.send(outputdata[i])

connectionSocket.send("\r\n".encode())

# Close the client connection socket

connectionSocket.close()

except IOError:

# Send HTTP response message for file not found

# Fill in start

connectionSocket.send("HTTP/1.1 404 Not found\r\n\r\n".encode())

# Fill in end

# Close the client connection socket

# Fill in start

connectionSocket.close()

# Fill in end

serverSocket.close()

Proxy server:

# Import socket module

from socket import \*

import requests

# create a cache

cache = []

# Create a TCP server socket

# (AF\_INET is used for IPv4 protocols)

# (SOCK\_STREAM is used for TCP)

serverSocket = socket(AF\_INET, SOCK\_STREAM)

# Defining serverSocket HOST and PORT, and listening

HOST = '127.0.0.1'

PORT = 32007

serverSocket.bind((HOST, PORT))

serverSocket.listen()

# Server should be up and running and listening to the incoming connections

while True:

print('Ready to serve...')

# Set up a new connection from the client

connectionSocket, addr = serverSocket.accept() # Fill in start #Fill in end

# If an exception occurs during the execution of try clause

# the rest of the clause is skipped

# If the exception type matches the word after except

# the except clause is executed

try:

# Receives the request message from the client

message = connectionSocket.recv(1024) # Fill in start #Fill in end

# Extract the path of the requested object from the message

# The path is the second part of HTTP header, identified by [1]

url = "http:/" + message.split()[1].decode()

cached = False

# Check if the url is cached, and if so return the data stored with code 200

for entry in cache:

if url == entry[0]:

cached = True

outputdata = entry[1]

# test print to let user know it was cached

print("returned from cache")

# Send the HTTP response header line to the connection socket

# Connection is successful if it gets here

connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n".encode())

# Send the content of the requested file to the connection socket

connectionSocket.sendall(outputdata)

# for i in range(0, len(outputdata)):

# connectionSocket.send(outputdata[i].encode())

connectionSocket.send("\r\n".encode())

# Close the client connection socket

connectionSocket.close()

# if data isn't cached ask server for data

if not cached:

# This sends an http request to the initial server

f = requests.get(url)

# check if status is OK

if f.status\_code == 200:

# Store the entire content of the requested file in a temporary buffer

outputdata = f.content

# add data to cache

cache.append((url, outputdata))

# Send the HTTP response header line to the connection socket

# Connection is successful if it gets here

connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n".encode())

# Send the content of the requested file to the connection socket

connectionSocket.sendall(outputdata)

# for i in range(0, len(outputdata)):

# connectionSocket.send(outputdata[i])

connectionSocket.send("\r\n".encode())

# Close the client connection socket

connectionSocket.close()

else:

# send 404 not found if status is not okay

connectionSocket.send("HTTP/1.1 404 Not found\r\n\r\n".encode())

# Close the client connection socket

connectionSocket.close()

except IOError:

# Send HTTP response message for file not found

connectionSocket.send("HTTP/1.1 404 Not found\r\n\r\n".encode())

# Close the client connection socket

connectionSocket.close()

serverSocket.close()