Lab 1: Web Server Lab

server.py

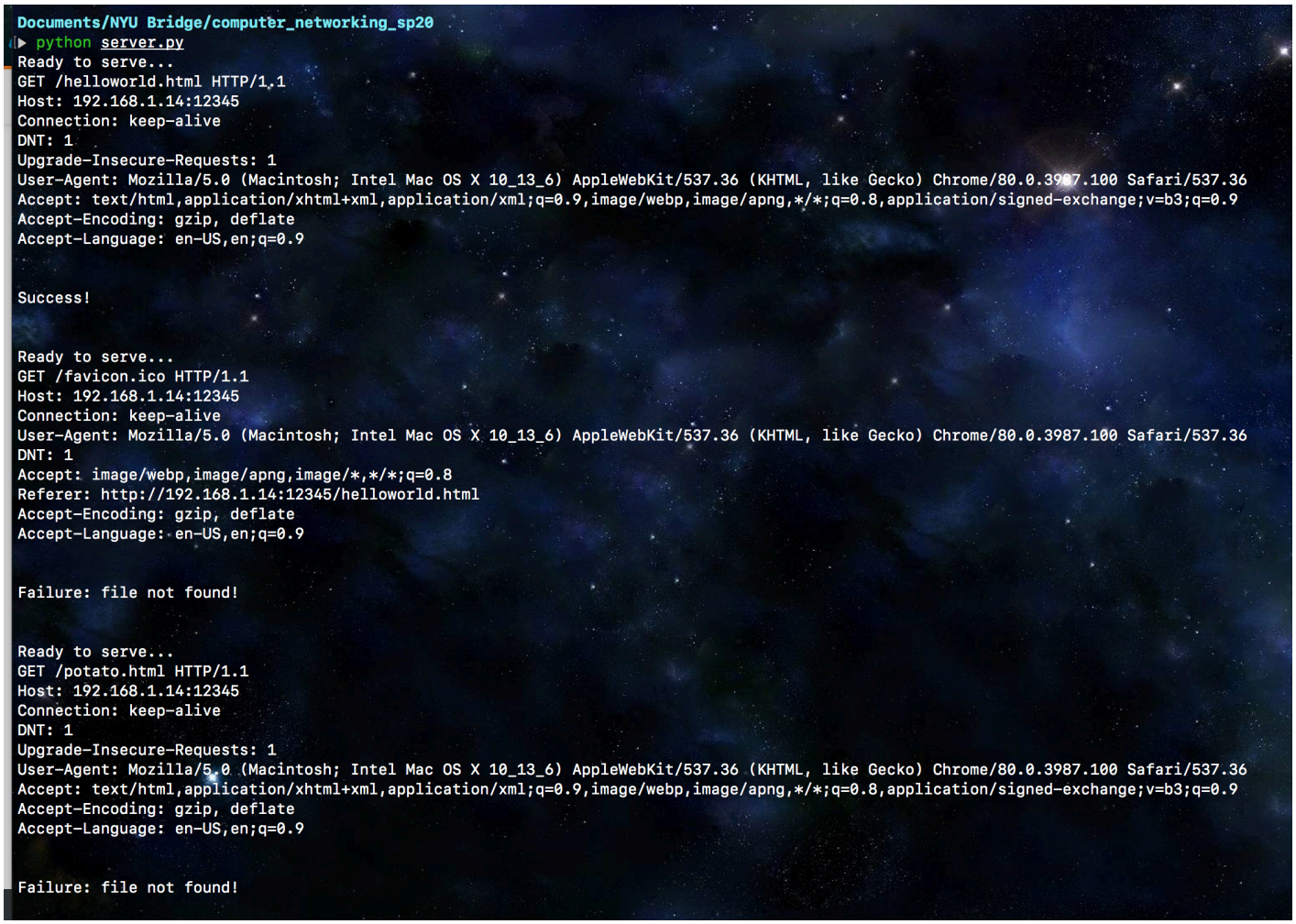
I hard-coded the port as 12345, the host is blank for localhost.

```
from socket import *
import sys

serverSocket = socket(AF_INET, SOCK_STREAM)

#Prepare a server socket
#Fill in start
serverPort = 12345
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
#Fill in end

while True:
    #Establish the connection
    print('Ready to serve...')
    connectionSocket, addr = serverSocket.accept() #Fill in start      #Fill in end
    try:
        message = connectionSocket.recv(1024)
        print(message)

        filename = message.split()[1]
        f = open(filename[1:])
        outputdata = f.read()

        #Send one HTTP header line into socket
        print("Success!\n\n")
        connectionSocket.send("\nHTTP/1.x 200 OK\n")

        #Send the content of the requested file to the client
        for i in range(0, len(outputdata)):
            connectionSocket.send(outputdata[i].encode())

        connectionSocket.send("\r\n".encode())
        connectionSocket.close()
    except IOError:
        #Send response message for file not found (404)
        print("Failure: file not found!\n\n")
        connectionSocket.send("\n404 File Not Found\n")
        connectionSocket.send("\r\n".encode())

        #Close client socket
        connectionSocket.close()

serverSocket.close()
sys.exit()  #Terminate the program after sending the corresponding data
```

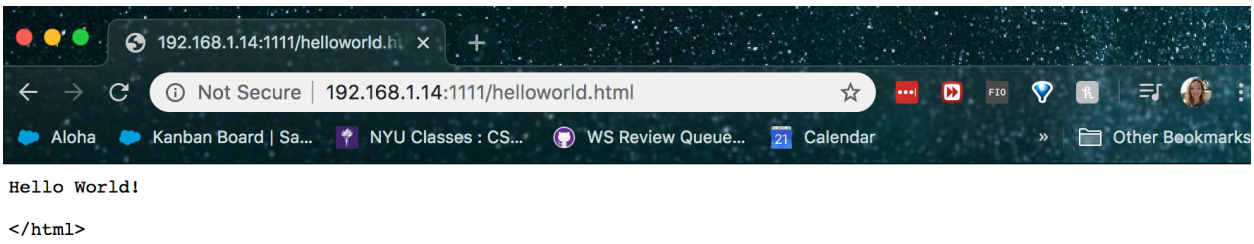
python server.py

First I put http://192.168.1.14:12345/helloworld.html in the browser, which received a 200 response, but a 404 for /favicon.ico because, well, we don't have a favicon file.

I then put http://192.168.1.14:12345/potato.html, which sent out 3 GET requests and received 3 404 responses. (Only the first fit in the screenshot.)



Browser – successful GET Request



The server doesn't send back any data in the event of a bad request, so you receive the browser's default "This page isn't working" when the GET request response code is not 200.

Optional Exercises 1

multithread_server.py – Again I hard-coded the port as 1111, & the host is default blank for localhost.

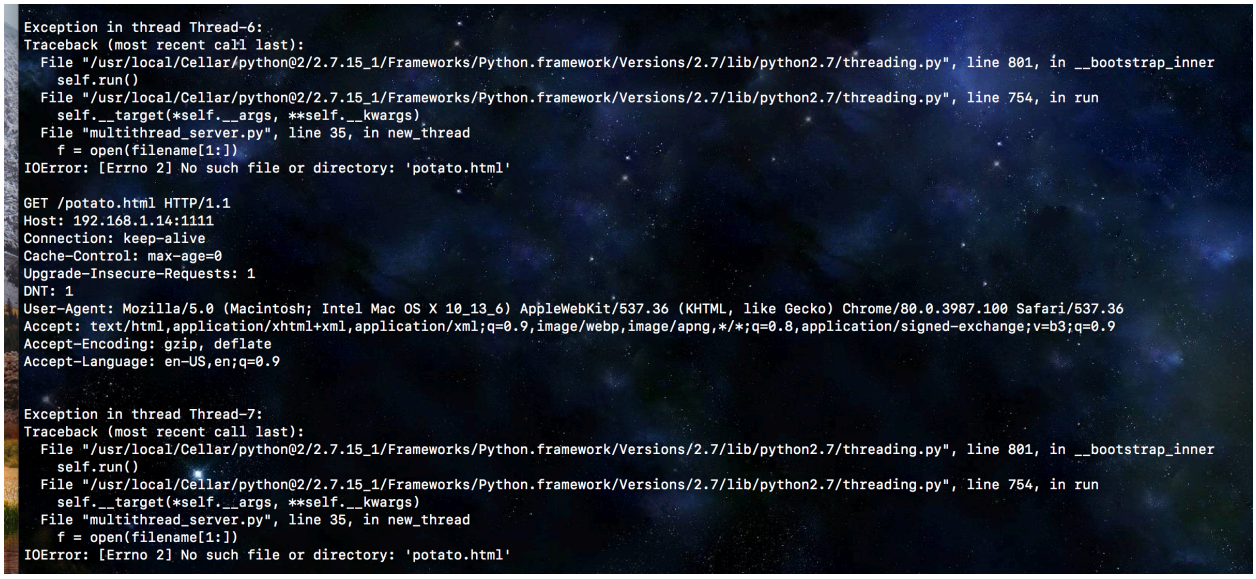
```
from socket import *
import threading
import sys

class ThreadedServer(object):
    def __init__(self):
        self.host = ''
        self.port = 1111
        self.serverSocket = socket(AF_INET, SOCK_STREAM)

    def run(self):
        print("Ready to serve...")

        self.serverSocket.bind((self.host, self.port))
        self.serverSocket.listen(10)

        while True:
            clientSocket, address = self.serverSocket.accept()
            clientSocket.settimeout(60)

            threading.Thread(
                target = self.new_thread,
                args = (clientSocket, address)
            ).start()

        serverSocket.close()

    def new_thread(self, clientSocket, address):
        while True:
            try:
                request = clientSocket.recv(1024)
                print(request)

                filename = request.split()[1]
                f = open(filename[1:])
                outputdata = f.read()
                clientSocket.send("\nHTTP/1.x 200 OK\n")

                for i in range(0, len(outputdata)):
                    clientSocket.send(outputdata[i].encode())

                clientSocket.send("\r\n".encode())
                clientSocket.close()
            except(error):
                print(error)
                clientSocket.send("\n400 Bad Request\n")
                clientSocket.close()
                sys.exit()

ThreadedServer().run()
```

python multithread_server.py

The browser page for calling http://192.168.1.14:12345/helloworld.html and http://192.168.1.14:12345/potato.html are the same as those of *server,py.*

As you can see below, there are multiple threads, with Exceptions thrown when the page does not exist. I opted to catch all errors and send a generic 400 response rather catch the specific IOError & throw a 404, even though in this lab we only encounter the 404 error to start off more generic; if I were writing a real program, I'd have an exception case for every common response code.

More threads error-ing out to show that multi-threading is working…



Optional Exercise 2

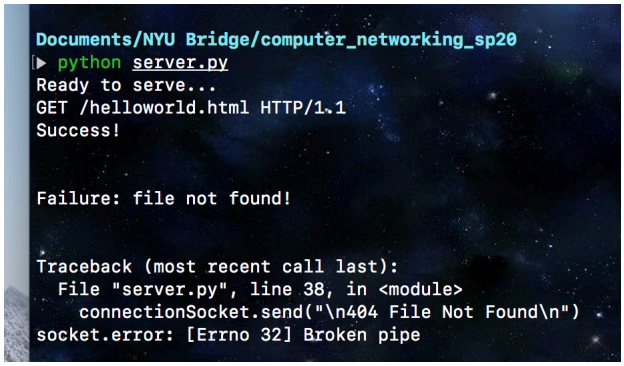
client.py

server.py.pdf | multithread_server.py | client.py | helloworld.html

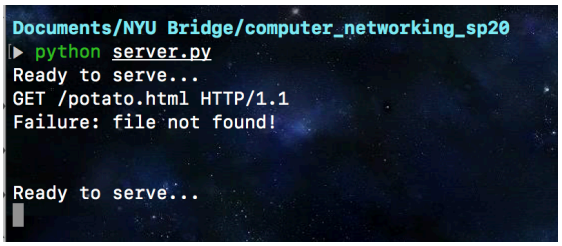
```python
from socket import *
import sys

clientSocket = socket(AF_INET, SOCK_STREAM)

if len(sys.argv) != 4:
    print("client: illegal command")
    print("usage: python3 client.py <server_host> <server_port> <file_name>")
    sys.exit()

host = str(sys.argv[1])
port = int(sys.argv[2])
request = str(sys.argv[3])
request = "GET /" + request + " HTTP/1.1"

try:
    clientSocket.connect((host,port))
    clientSocket.send(request)

    response = clientSocket.recv(1024)
    print(response)
except(error, data):
    print(error, data)
    print("Something went wrong.")
    sys.exit()

clientSocket.close()

```

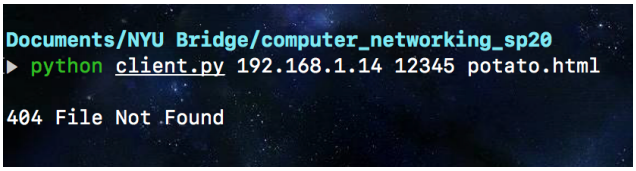
A simple client that gets the *host*, *port*, and *file_name* from the CLI and makes requests to the server. It doesn't automatically retry requests upon 400 responses. In the event of an error, the client closes the connection with an opaque error message.

As you can see, GET /helloworld.html is a success, but the server shuts down due to the 404 error from the favicon, causing the GET /potato.html to get a socket error.



Let’s fire it up again…





This time the server returns a 404 response and continues waiting for requests. The client program exits.