University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 2 : Introduction to Socket Programming

Submitted By:

Name: Muztoba Hasan Sinha

Roll No : 15

Name: Bholanath Das Niloy

Roll No : 22

Submitted On :

January 25^{th}, 2023

Submitted To :

Dr. Md. Abdur Razzaque

Md Mahmudur Rahman

Md. Ashraful Islam

Md. Fahim Arefin

Contents

1 Introduction

In this lab we have we were given a problem to demonstrate our capabilities in understanding socket programming and to use it to solve the basic task of connecting 2 computers, one as a client the other as a server.

2 Objectives

Creating TCP Connections using Socket Programming

1. Establish a TCP connection between a server process running on host A and client process running on Host B and complete these operations requested from host A.
   (a) Small letter to capital
   (b) Checking whether a number is prime or not
2. Using the above connection, design and implement a non-idempotent operation using exactly- once semantics that can handle failure of request messages, failure of response messages and process execution failures.
   (a) Design and describe an application-level protocol to be used between an automatic teller machine and a bank's centralized server. Your protocol should allow a user's card and password to be verified, the account balance (which is maintained at the centralized computer) to be queried, and an account withdrawal to be made (that is, money disbursed to the user). Your protocol entities should be able to handle the all-too-common cases in which there is not enough money in the account to cover the withdrawal. Specify your protocol by listing the messages exchanged and the action taken by the automatic teller machine or the bank's centralized computer on transmission and receipt of messages. Sketch the operation of your protocol for the case of a simple withdrawal with no errors.
   (b) HOME WORK – Enhance the above protocol so that it can handle errors related to both request and response messages to and from the server.

3 Methodology

3.1 Code

We have coded all of this in python as is shown below.

1. Converting string from Upper to Lower Case
   (a) Client Side Code

```python
import socket

def client_program():
    host = '10.33.2.77'
    port = 5000

    client_socket = socket.socket()
    client_socket.connect((host, port))

    message = input(" -> ")
    while message.lower().strip() != 'end':
        client_socket.send(message.encode())
        data = client_socket.recv(1024).decode()

        if not data:
            break

        print(data)

        message = input(" -> ")

    client_socket.close()
```

```
23
24 if __name__ == '__main__':
25     client_program()
```

Listing 1: Client Side Code

(b) Server Side Code

```
1  import socket
2
3  def server_program():
4      host = ""
5      port = 5000
6
7      server_socket = socket.socket()
8      server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
9
10     server_socket.bind((host, port))
11
12     server_socket.listen(2)
13     conn, address = server_socket.accept()
14     print("Connection from: " + str(address))
15     while True:
16         data = conn.recv(1024).decode()
17         if not data:
18             break
19         print("from connected user: " + str(data))
20         conn.send(data.lower().encode())
21     conn.close()
22
23 if __name__ == '__main__':
24     server_program()
```

Listing 2: Server Side Code

2. Checking if a number is prime or not

(a) Client Side Code

```
1  import socket
2
3  def client_program():
4      host = '10.33.2.77'
5      port = 5000
6
7      client_socket = socket.socket()
8      client_socket.connect((host, port))
9
10     message = input(" -> ")
11     while message.lower().strip() != 'bye':
12         client_socket.send(message.encode())
13         data = client_socket.recv(1024).decode()
14
15         print('Received from server: ' + data)
16
17         message = input(" -> ")
18
19     client_socket.close()
20
21 if __name__ == '__main__':
22     client_program()
```

Listing 3: Client Side Code for Prime Checking

(b) Server Side Code

```
1  import socket
2
3  def is_prime(n):
4      for i in range(2,n):
5          if (n%i) == 0:
```

```
            return False
    return True

def server_program():
    host = ""
    port = 5000

    server_socket = socket.socket()
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    server_socket.bind((host, port))

    server_socket.listen(2)
    conn, address = server_socket.accept()
    print("Connection from: " + str(address))
    while True:
        data = conn.recv(1024).decode()
        if not data:
            break
        res = 'Not a prime'
        try:
            x = int(data)
            if is_prime(x): res = 'Is a Prime'
        except:
            res = 'Not an integer number'
            pass
        print("from connected user: " + str(data))
        conn.send(res.encode())

    conn.close()
if __name__ == '__main__':
    server_program()
```

Listing 4: Server Side Code for Prime Checking

3. Coding up an ATM machine to withdraw/deposit money from

   (a) Client Side Code

```
import socket
import time
import threading
from threading import Timer
import random

THRESHOLD = 10
SERVER = socket.gethostbyname(socket.gethostname())
PORT = 5000
ADDR = (SERVER, PORT)

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)

class RepeatTimer(Timer):
    def run(self):
        while not self.finished.wait(self.interval):
            self.function(*self.args, **self.kwargs)

def send(msg):
    message = msg.encode()
    client.send(message)


while True:
    data = client.recv(1024).decode()
    if data:
        if "requestId" in data:
            print(data)
            rID = data.partition("requestId:")[2]
            inp = input('->')
```

```
            req = f"{rID},{inp}"
            t = time.process_time()
            sendTd = RepeatTimer(1, send, [req])
            sendTd.start()
            while True:
                data = client.recv(1024).decode()
                if data:
                    rando = random.randint(0,99)
                    if rando > THRESHOLD:
                        continue
                    sendTd.cancel()
                    print(data)
                    elapsed_time = time.process_time()-t
                    print(f"elapsed time: {elapsed_time}")
                    break
            continue
        if ("Logged" in data) or ("Withdrawn" in data) or ("Deposited" in data) or (("Enter" in data)):
            print(data+"\n")
            continue
        else:
            print(data)
            inp = input('->')
            send(inp)
```

Listing 5: Client Side Code for ATM

(b) Server Side Code

```
import socket
import random


users = {
    1:"password",
    2:"password2"
}
taka = {
    1: 1000,
    2: 2000
}
requests = {
    0: ""
}


THRESHOLD = 10
PORT = 5000
SERVER = socket.gethostbyname(socket.gethostname())
ADDR = (SERVER, PORT)


server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind(ADDR)

def send(conn, msg):
    conn.send(msg.encode())


def start():
    server.listen()
    print(f"server is listening on {SERVER}, PORT::{PORT}")
    conn, addr = server.accept()
    connected = True
    print(f"{addr} connected.")
    logged = False
    useridInp = False
    prompted = False
    optSel = False
    while connected:
```

```
        print(useridInp, logged, prompted, optSel)
        if useridInp == False:
            send(conn, "Please enter your UserID")
        if logged == False or prompted == True:
            data = conn.recv(1024).decode()

        if data:
            if "," in data:
                req_data = data.split(",")
                if int(req_data[0]) in requests.keys():
                    send(conn, requests[int(req_data[0])])
                else :
                    data = req_data[1]
            if data == "bye":
                connected = False
                print(f"{addr} disconnected.")
                conn.close()
            print(f"user {addr} input {data}")

            if useridInp == False:
                id = int(data)
                print(data)
                if id in users.keys():
                    useridInp = True
                else:
                    send(conn, "sorry wrong userId")
                send(conn, "pls type pass")
                continue

            if logged == False:
                print(data)
                if users[id] == data:
                    logged = True
                    send(conn, "Logged In\n")
                else:
                    send(conn, "sorry wrong password")
                continue

            if prompted == False:
                prompt = f"""You have {taka[id]} in account.
                What do you want to do?
                (1) withdraw money
                (2) deposit money
                (bye) exit
                """
                send(conn, prompt)
                prompted = True
                print("prompt for withdraw")
                continue

            if optSel == False:
                requestID = max(k for k, v in requests.items()) + 1
                optSel = True
                opt = int(data)
                if opt == 1:
                    send(conn, "Enter amount to withdraw ")
                    send(conn, f"requestId: {requestID}")
                else :
                    send(conn, "Enter amount to deposit")
                    send(conn, f" requestId: {requestID}")
                continue
            optSel = False
            prompted = False

            if opt == 1:
                amount = float(data)
                if amount <= taka[id]:
                    taka[id] -= amount
                    confirmation = f"{amount} Withdrawn. Current Balance {taka[id]}
    "
```

```
                    requests.update({requestID : confirmation})
                    rando = random.randint(0,99)
                    if rando > THRESHOLD:
                        continue
                    send(conn, confirmation)
                else:
                    confirmation = "insufficeint funds"
                    send(conn, confirmation)
                    rando = random.randint(0,99)
                    if rando > THRESHOLD:
                        continue
                    requests.update({requestID : confirmation})
            else:
                amount = float(data)
                taka[id] += amount
                confirmation = f"{amount} Deposited. Current Balance {taka[id]}"
                requests.update({requestID : confirmation})
                rando = random.randint(0,99)
                if rando > THRESHOLD:
                    continue
                send(conn, confirmation)
    conn.close()

print("starting server")
start()
```

Listing 6: Server Side Code for ATM

3.2 Theory

This required an immense amount of effort to do correctly and implement the error methods and the time elapsed results. This was done using timer in python and different threads.

3.2.1 Socket Programming In Python

Server side We have to declare a socket by specifying it's family and kind. We declare a pair of the server ip and the port number and bind it to the socket for the server side. We start listening on the port from the server. We accept connections and map the connection and client address to variables for server side use. We have declared a custom function to send data to client using the connection previously saved. Client Side We have to declare a socket by specifying it's family and kind. We make an address tuple by specifying the server address and port. We connect to the server using this. The custom send function is used here too.

3.2.2 Client Side

What we are basically doing is taking a hostname for our server and connecting the client with the server. There are some major components to the client side code, they are as follows:

1. **Repeat-Timer**
   We have made a custom timer for our use case which is running on another thread and is the main reason we are able to re-transmit the queries if for some reason the connection ends up dropping.

2. **Send Messages**
   This basically just uses the established TCP connection and sends messages to the server for it process them;

3. **Main Loop**
   This is a infinite loop which receives data at the start. Then we check for the validity of the data. Then we filter if it is a request then try to resend the requests until we receive a confirmation on the client side. This ensures that the client side always send the most important request packet and also receives confirmation for it. Using this we have handled both server-side and client-side error transmissions in one go.

3.2.3 Server Side

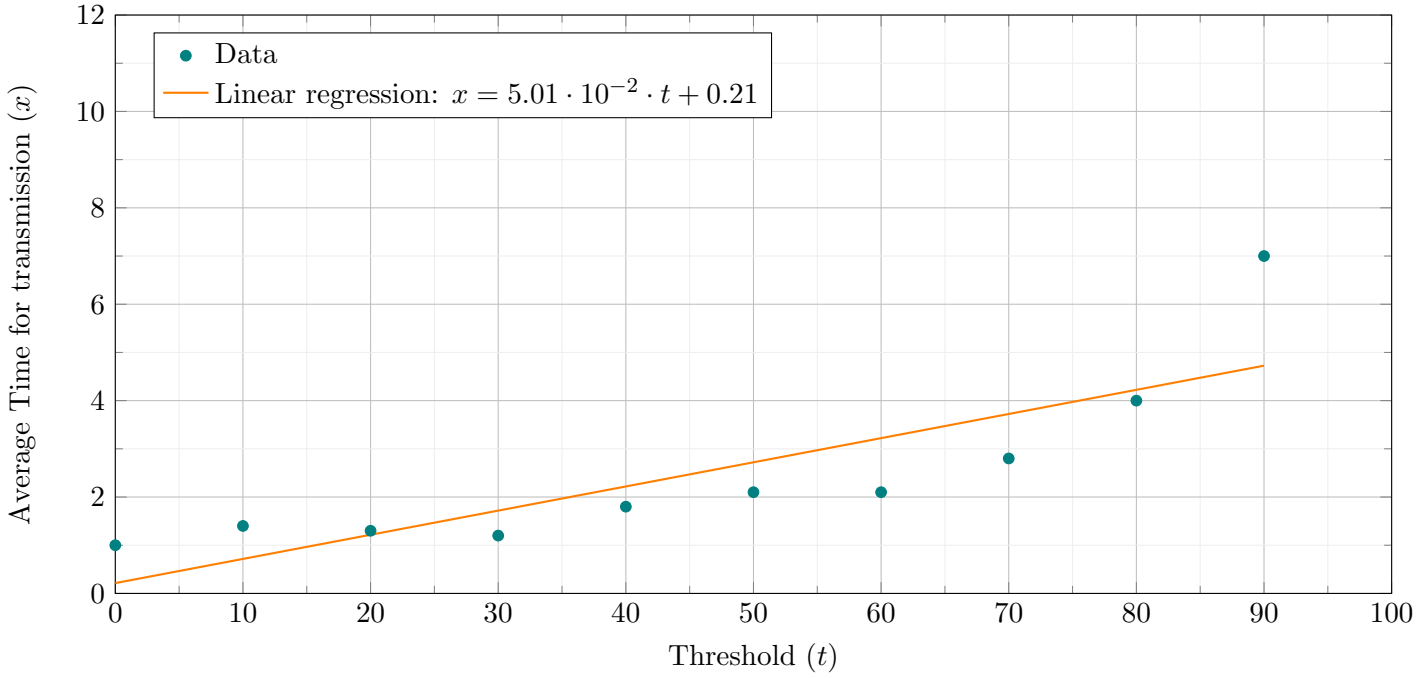
A few major components of the serverside is described as follows

1. **Dictionaries**
   We have saved the user data and the request queues on dictionaries for easy lookup. In future we could load/store these from files.

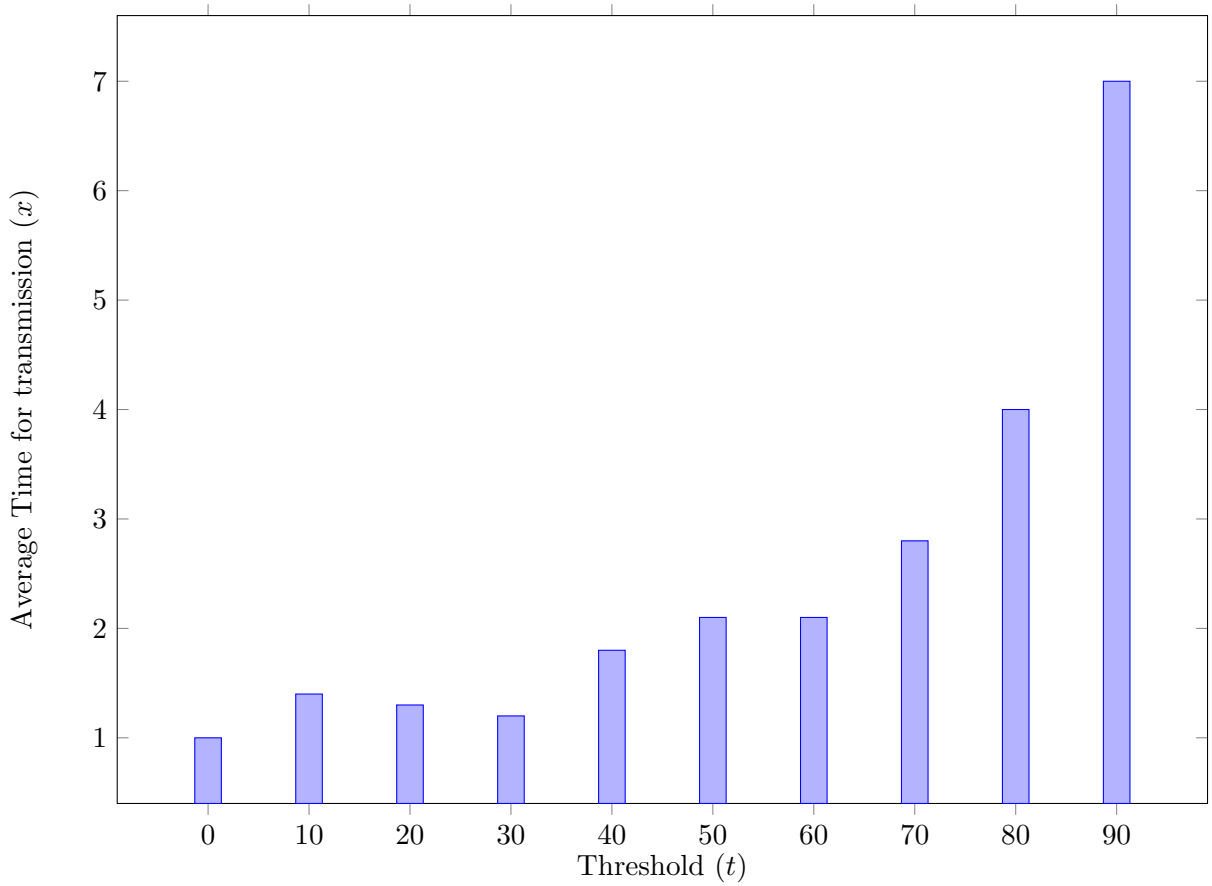
2. **Start**
   This handles the business logic of the server. It receives data at the start and funnels it through the decision tree as needed. We have utilized quite a few variables to guide the flow and check each received string at the beginning to see if it is a processed request or not.

4 Experimental Results

4.1 Reconnection-Time Graphs



As we can see this does not fit a straight line at all and as simple logic would imply it would become infinitely harder to reach a certain threshold and thus gives somewhat of an intuition as to why this function would grow exponentially.



4.2 Screenshots of Working Code

1. Screenshot for the Upper to Lower Case

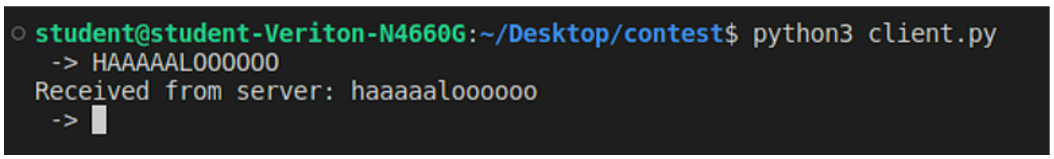


Figure 1: Upper to Lower

2. Screenshot for the Prime Check

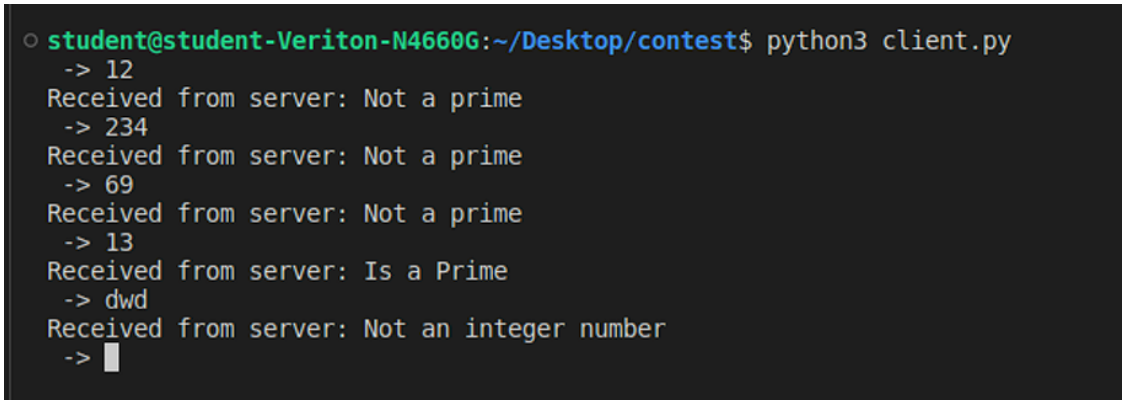


Figure 2: Prime Check

3. Screenshot of the ATM working

```
❯ python3 server_fin.py
starting server
server is listening on 127.0.1.1, PORT::5000
('127.0.0.1', 40076) connected.
False False False False
user ('127.0.0.1', 40076) input 1
1
True False False False
user ('127.0.0.1', 40076) input password
password
True True False False
user ('127.0.0.1', 40076) input password
prompt for withdraw
True True True False
user ('127.0.0.1', 40076) input 1
True True True True
user ('127.0.0.1', 40076) input 100
True True False False
user ('127.0.0.1', 40076) input 100
prompt for withdraw
True True True False
_
```

```
❯ python3 client_fin.py
Please enter your UserID
->1
pls type pass
->password
Logged In


You have 1000 in account.
                What do you want to do?
                (1) withdraw money
                (2) deposit money
                (bye) exit

->1
Enter amount to withdraw

requestId: 1
->100
100.0 Withdrawn. Current Balance 900.0
elapsed time: 1.001089096069336
You have 900.0 in account.
                What do you want to do?
                (1) withdraw money
                (2) deposit money
                (bye) exit

->
```

Figure 3: ATM

5 Experience

1. Learnt about socket programming in python.
2. Learned to work with threading.
3. Preliminary ideas about server side programming.
4. Learnt graphing in LaTeX

References

[1] Socket programming in python. *GeeksforGeeks*, jun 20 2017. [Online; accessed 2023-01-25].

[2] Pankaj. Python socket programming - Server, client example. *DigitalOcean*, aug 3 2022. [Online; accessed 2023-01-25].

[3] Real Python. Python timer functions: Three ways to monitor your code. *Real Python*, mar 21 2022. [Online; accessed 2023-01-25].

[4] Real Python. Socket programming in python (guide). *Real Python*, feb 21 2022. [Online; accessed 2023-01-25].