EECS563 Socket Programming Assignment

Name: Stephen Longofono
Date: September 19, 2017

Required & Extra Credit Questions

1. What are example applications using TCP and UDP? Give two examples for each protocol. What are the port numbers these applications use?

   For TCP connections, common applications include SSH (secure shell) and HTTP/Web traffic. HTTP traffic is typically conducted on port 80 and SSH traffic is typically conducted on port 22.

   For UDP, common applications include VXLAN (Virtual eXtensible LAN), and DNS (domain name server). VXLAN traffic is typically conducted on port 4789 and DNS traffic is typically conducted on port 53. Note that DNA is also used over TCP connections, but for high-volume DNS servers, the low overhead of UDP is preferable.

2. In the given example, why does the UDP server need only one socket, whereas the TCP server needs two sockets? If the TCP server needs to support n simultaneous connections, each from a different client host, how many sockets would the TCP server need?

   TCP servers serve multiple clients effectively by maintaining a "welcoming socket," with a publicly known IP address. When a client initiates a connection, the server then creates another socket specifically for that client (uniquely identified by the client IP and port). In contrast, a UDP server creates a single socket and waits for a connection request. The UDP server completes the three-way handshake and communicates with the same socket, so it needs only one.

   If a TCP server needed to suppot n unique clients, it needs $n + 1$ sockets: one for communicating with each of the clients, and a "welcoming socket" to accept and establish connections.

3. Suppose we add the following line to the UDP client program on line 17, after creating the socket: *clientSocket.bind((", 5434))* Will it be necessary to change the UDP server program? What are the port numbers for the sockets in the UDP client and server programs? What were the port numbers before making this change?

   After making those changes, it is not necessary to change the server program. Binding the client socket to a specific port will ensure that the selected port gets used, but as far as the server is concerned, it receives some input from an IP address with some port, and it sends its reply to that IP address and port. If those changes were made, we could be sure that the client is bound to 5434, and the server is bound to the port we specified when we brought it up (say, 5050). Without the changes, we

can know the server port (again, say 5050 as specified when we run the server), but the client port may change. The only way to know is to explicitly request the port number after creating the client UDP socket via the *getsockname()* function.

Code, TCP Client Application

```
"""
client_TCP.py

This file demonstrates a simple TCP client which sends data to a remote server
and gathers the reply.

Usage:
    python client_TCP.py --help
"""

import socket
import argparse

# Trims input to expected maximum size
def trim(myStr):
    if len(myStr) > 2048:
        print("Your input was larger than the maximum of {} bytes and has"
              "been truncated to fit.".format(BUFLEN))
        return myStr[:2048-len(myStr)]
    return myStr

# Per instructions, "print the address and port using a function"
def print_address_port(a,b):
    print("\tIP Address {}\n\tPort {}\n".format(a,b))

BUFLEN = 2048

parser = argparse.ArgumentParser()
parser.add_argument('--port',
                    '-P',
                    type=int,
                    help='The desired port on the remote server',
                    required=True)
parser.add_argument('--address',
                    '-A',
                    type=str,
                    help='The IP address of the remote server to connect to',
                    required=True)
```

```
args = parser.parse_args()

PORT = args.port
SERVER_ADDR = args.address

print("Attempting to connect to IP address {}"
      "on port {}...".format(SERVER_ADDR, PORT))

try:
    sock = None

    # Set up connection
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((SERVER_ADDR, PORT))

    print("Connection established to IP address {}".format(SERVER_ADDR))
    print("Press Ctrl-C to quit")

    while True:

        raw = raw_input("Please enter the statement: ")

        # Send the user's input through the socket TCP connection
        sock.send(trim(raw))

        print("Local details:\n")
        localaddress, localport = sock.getsockname()
        print_address_port(localaddress, localport)

        # Grab up to BUFLEN bytes from the connection reply
        payload = sock.recv(BUFLEN)
        if payload:
            print("Return text from the server: {}".format(payload))
            print("Remote details:\n")
            print_address_port(SERVER_ADDR, PORT)


except KeyboardInterrupt:
    # Handle intentional quit gracefully
    print("\n\nQuitting...")

finally:
    # Tear-down connection
    if sock is not None:
        sock.close()
```

Code, TCP Server Application

```
"""
server_TCP.py

This file demonstrates a simple TCP server which accepts strings and returns
the string with all lowercase characters replaced by their uppercase
characters.

Usage:
    python server_TCP.py -P <Port to listen on>
"""

import socket
import argparse

# Converts string argument to their uppercase representation
def convert(myStr):
    if(type(myStr) != str):
        raise RuntimeException(
        'Cannot convert non-string input to lower case.'
        ' Please pass in a string')
    return myStr.upper()

# Per instructions, "print the address and port using a function"
def print_address_port(a,b):
    print("\tIP Address {}\n\tPort {}\n".format(a,b))

IP_ADDR = '127.0.0.1' # Use localhost
BUFLEN = 2048

parser = argparse.ArgumentParser()
parser.add_argument('--port',
                    '-P',
                    type=int,
                    help='The desired port for socket communication',
                    required=True)
args = parser.parse_args()

PORT = args.port

print("Listening at IP address {} on port {}...".format(IP_ADDR, PORT))

try:

    connection = None
```

```
    # Set up connection
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind((IP_ADDR, PORT))
    sock.listen(1) # listen for a maximum of 1 connections

    # If we got this far, we found someone trying to connect.  Accept the
    # connection and gather the connection object and its IP address.
    connection, client_address = sock.accept()


    print("Connection established to IP address {}".format(client_address))
    print("Local details:\n")
    localaddr, localport = connection.getsockname()
    print_address_port(localaddr, localport)

    while True:
        # Grab up to BUFLEN bytes from the connection
        payload = connection.recv(BUFLEN)
        if payload:
            print("Received text from client: {}".format(payload))
            print("Remote details:\n")
            print_address_port(client_address[0], client_address[1])

            # Send back the altered text
            connection.send(convert(payload))

except KeyboardInterrupt:
    # Handle intentional quit gracefully
    print("\n\nQuitting...")

finally:
    # Tear-down connection
    if connection is not None:
        connection.close()
```

Code, UDP Client Application

```
"""
client_UDP.py

This file demonstrates a simple UDP client which sends data to a remote server
and gathers the reply.

Usage:
    python client_UDP.py --help
```

```
"""

import socket
import argparse

# Trims input to expected maximum size
def trim(myStr):
    if len(myStr) > 2048:
        print("Your input was larger than the maximum of {} bytes and has "
              "been truncated to fit.".format(BUFLEN))
        return myStr[:2048-len(myStr)]
    return myStr

# Per instructions, "use a function to print address and port"
def print_address_port(a,b):
    print("\tIP Address {}\n\tPort {}\n".format(a,b))

BUFLEN = 2048

parser = argparse.ArgumentParser()
parser.add_argument('--port',
                    '-P',
                    type=int,
                    help='The desired port on the remote server',
                    required=True)
parser.add_argument('--address',
                    '-A',
                    type=str,
                    help='The IP address of the remote server to connect to',
                    required=True)

args = parser.parse_args()

PORT = args.port
SERVER_ADDR = args.address

try:
    # Establish a UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    print("Local socket created")
    print("Press Ctrl-C to quit")

    while True:

        raw = raw_input("Please enter the statement: ")
```

```
            # Send the user's input through the socket TCP connection
            sock.sendto(trim(raw), (SERVER_ADDR, PORT))

            print("Sent message to remote server")

            # Use built-in functions to determine what port we ended up using
            print("Local details:\n")
            print(sock.getsockname())
            localaddr, localport = sock.getsockname()
            print_address_port(localaddr, localport)


            # Grab up to BUFLEN bytes from the connection reply
            # In this case, we don't specify the port, we simply look for any
            # response.  This doesn't seem safe.
            payload, address = sock.recvfrom(BUFLEN)
            if None != payload:
                print("Return text from the server: {}".format(payload))
                print("Remote server details:\n")
                print_address_port(address, PORT)

    except KeyboardInterrupt:
        # Handle intentional quit gracefully
        print("\n\nQuitting...")
```

Code, UDP Server Application

```
"""
server_UDP.py

This file demonstrates a simple UDP server which accepts strings and returns
the string with all lowercase characters replaced by their uppercase
characters.

Usage:
    python server_UDP.py -P <Port to listen on>
"""

import socket
import argparse

# Converts string argument to their uppercase representation
def convert(myStr):
    if(type(myStr) != str):
        return str(myStr).upper()
    return myStr.upper()
```

```
# Per instructions, "use a function to print address and port"
def print_address_port(a,b):
    print("\tIP Address {}\n\tPort {}\n".format(a,b))

BUFLEN = 2048

parser = argparse.ArgumentParser()
parser.add_argument('--port',
                    '-P',
                    type=int,
                    help='The desired port to listen on',
                    required=True)

args = parser.parse_args()

PORT = args.port

try:
    # Set up connection, use internet protocol and datagrams (UDP)
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind(('', PORT))
    print("Local socket created")
    print("Press Ctrl-C to quit")


    while True:
        # Grab up to BUFLEN bytes from the connection
        payload, address = sock.recvfrom(BUFLEN)

        if payload:
            print("Local details:\n")
            localaddr, localport = sock.getsockname()
            print_address_port(localaddr, localport)

            print("Received {} from a client...".format(payload))
            print("Remote details:\n")
            print_address_port(address[0], address[1])

            # Send back the altered text
            # We can't use the same port we are listening on, since it will
            # create an endless loop.  Instead, we simply send it to the
            # address we received the message from, and let the other end
            # decide what to do with it.
            sock.sendto(convert(payload), address)
```

```
except KeyboardInterrupt:
    # Handle intentional quit gracefully
    print("\n\nQuitting...")
```

Figure 1: Input & Output for the Client Applications

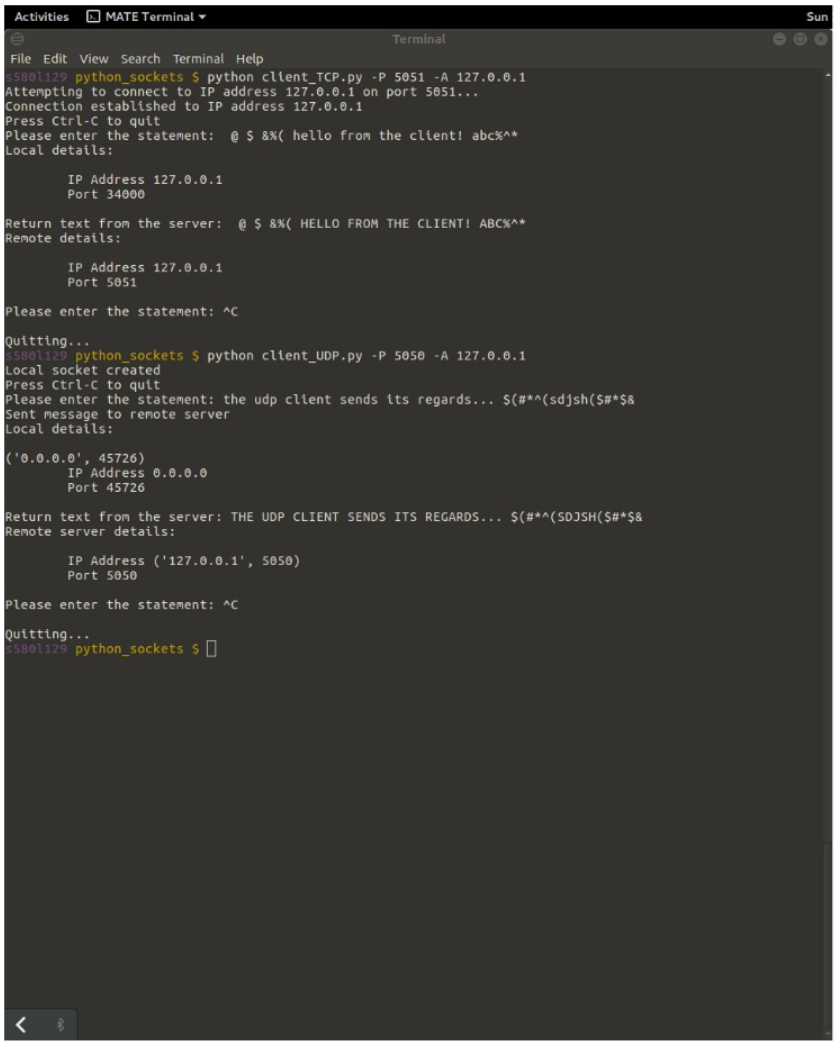

Figure 2: Input & Output for the Server Applications