

Programming Assignment 3

CST 311-30, Introduction to Computer Networks, Online

READ INSTRUCTIONS CAREFULLY BEFORE YOU START THE ASSIGNMENT.

Assignment must be submitted electronically as a pdf file to Canvas by 11:59 p.m. on the due date. Late assignments will not be accepted.

This assignment is to be done with your Team per the Programming Process document.

The assignment requires you to submit both client and server programs in a zip file. The naming convention of the file should be PA3_your_team_number.zip.

Put your names in the program as well. Your client and server programs must meet the requirements below. Your program must have sufficient comments to clearly explain how your code works. Your code must compile to get partial credit.

This assignment is worth 150 points.

Purpose

The purpose of this assignment is to satisfy one of the stated outcomes in the syllabus:

- b. Develop simple software programs using sockets to achieve communication between two (or more) computers.

You will write the program in Python where the interface to the TCP/IP application programming interface is similar to other C-family programming languages.

Context

The Transmission Control Protocol(TCP) allows for connection-oriented sessions between clients and a server. This means that multiple clients can connect to a single server and the communication is managed in separate sessions for each client. The initial connection setup is done once and then packets are exchanged until the connection is closed. A common pattern is for a TCP server to start up and allow multiple connections; in fact, this is exactly how HTTP web servers work.

There are an endless number of applications that rely on the kind of communication service that you will develop in this assignment, here are a few examples:

- HTTP web server

- A chat service, such as Google Hangouts or Slack
- Data delivery service, such as the GPS position data in the <http://odss.mbari.org> backend

Many successful businesses have been built by building rich applications on top of a TCP/IP service such as the one you will create in this assignment.

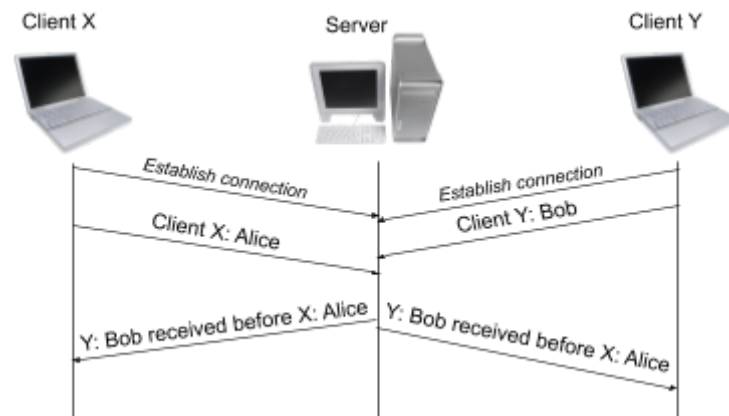
Task

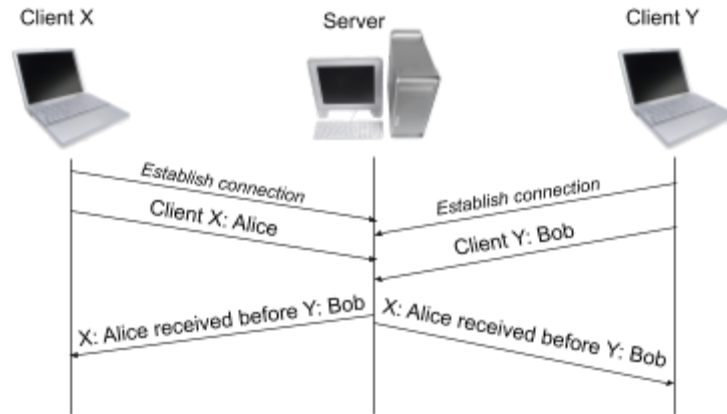
Your task is to write client code that satisfies the following requirements:

Client code

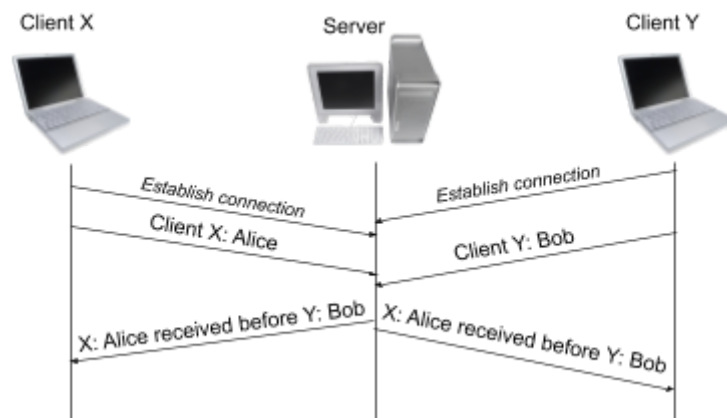
There are two clients, X and Y (both essentially identical) that will communicate with a server. Clients X and Y will each open a TCP socket to your server and wait until the server establishes a connection with both the clients. In the next step, one of the clients sends a message to your server followed by the other client.. The message contains the name of the client followed by a name (e.g., “Client X: Alice”, “Client Y: Bob”).

Later clients receive a message back from the server, indicating which message arrived at the Server first and which arrived second. The clients should print the message that they sent to the server, followed by the reply received from the server. The following figures explain the problem.





The response message does not rely on the order of the connection establishment. If the connections are established in a different order the response will still be dependent only on the order of the messages received at the server.



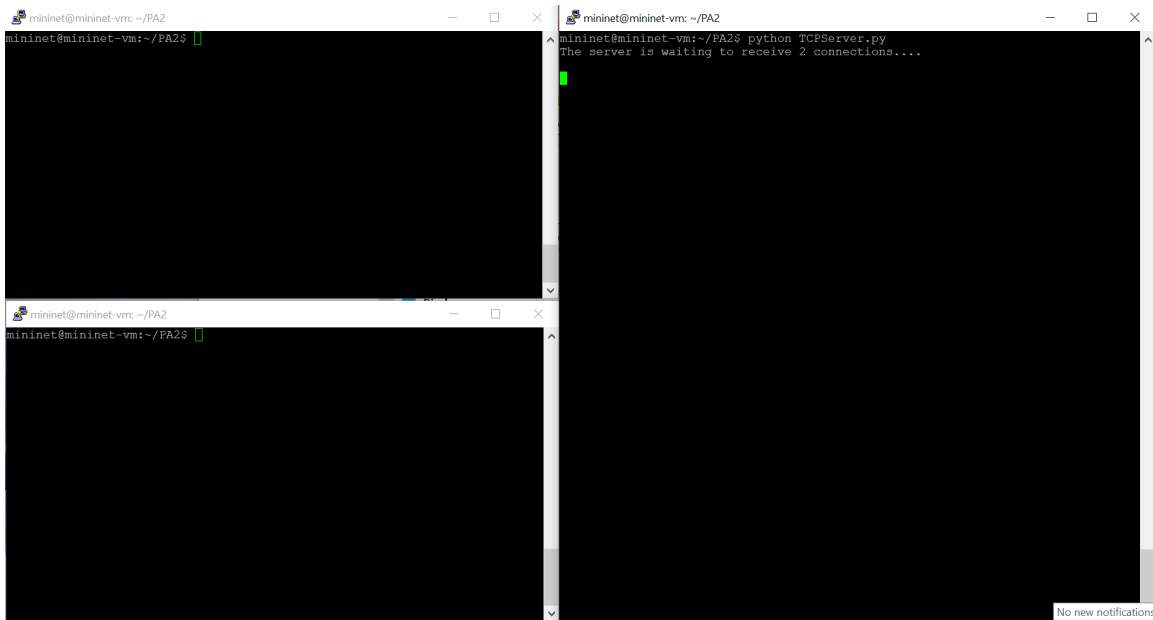
Server code

The server will accept connections from both clients and after it has received messages from both X and Y, the server will print their messages and then send an acknowledgment back to your clients. The acknowledgment from the server should contain the sequence in which the client messages were received (“X: Alice received before Y: Bob”, or “Y: Bob received before X: Alice”). After the server sends out this message it should output a message saying - “Sent acknowledgment to both X and Y”. Your server can then terminate.

The server sits in an infinite loop listening for incoming TCP packets. When a packet comes, the server simply sends it back to the client. You can use the TCP server/client programs from the previous programming assignment as templates to start and then modify it to build your programming assignment.

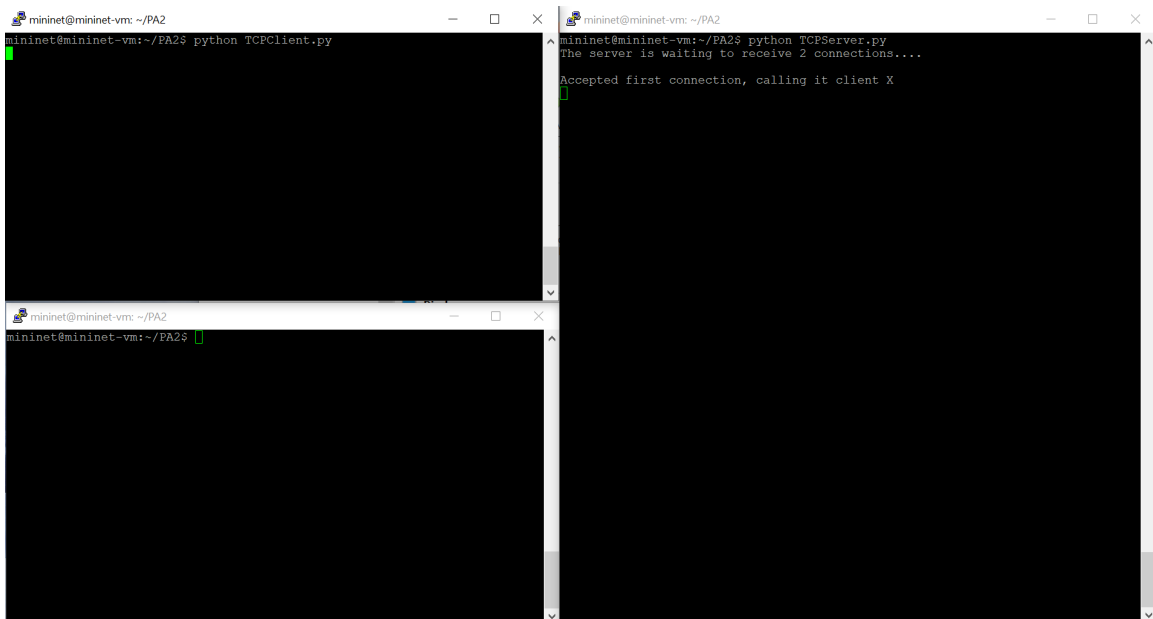
Expected output

Server is started



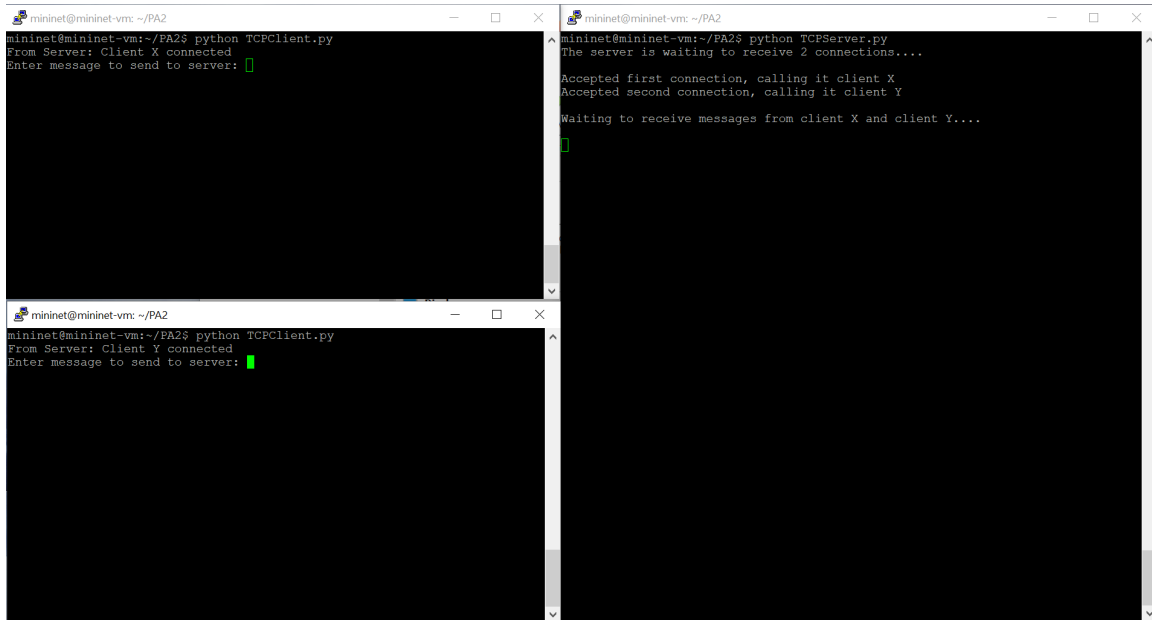
The screenshot shows two terminal windows. The top-left window is titled 'mininet@mininet-vm: ~/PA2' and shows a green cursor on the prompt 'mininet@mininet-vm:~/PA2\$'. The top-right window is also titled 'mininet@mininet-vm: ~/PA2' and shows the command 'python TCPServer.py' being executed, with the output 'The server is waiting to receive 2 connections....'. The bottom-left window is titled 'mininet@mininet-vm: ~/PA2' and shows a green cursor on the prompt 'mininet@mininet-vm:~/PA2\$'. The bottom-right window is titled 'mininet@mininet-vm: ~/PA2' and shows a green cursor on the prompt 'mininet@mininet-vm:~/PA2\$'. A 'No new notifications' banner is visible at the bottom right of the terminal area.

One client requests a connection and is accepted at the server (server calls it X)....



The screenshot shows two terminal windows. The top-left window is titled 'mininet@mininet-vm: ~/PA2' and shows the command 'python TCPClient.py' being executed, with a green cursor on the prompt 'mininet@mininet-vm:~/PA2\$'. The top-right window is titled 'mininet@mininet-vm: ~/PA2' and shows the output 'Accepted first connection, calling it client X' from the server. The bottom-left window is titled 'mininet@mininet-vm: ~/PA2' and shows a green cursor on the prompt 'mininet@mininet-vm:~/PA2\$'. The bottom-right window is titled 'mininet@mininet-vm: ~/PA2' and shows a green cursor on the prompt 'mininet@mininet-vm:~/PA2\$'.

Second client requests a connection and is accepted at the server (server calls it Y)



```
mininet@mininet-vm: ~/PA2
mininet@mininet-vm:~/PA2$ python TCPCClient.py
From Server: Client X connected
Enter message to send to server:

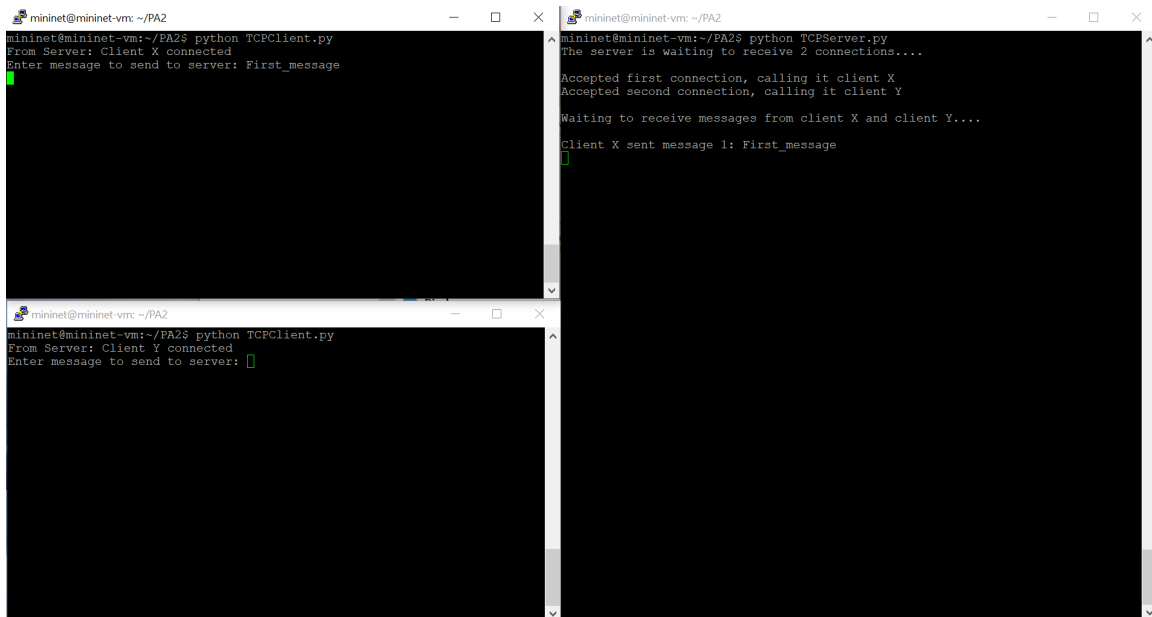
mininet@mininet-vm: ~/PA2
mininet@mininet-vm:~/PA2$ python TCPCClient.py
From Server: Client Y connected
Enter message to send to server:

mininet@mininet-vm: ~/PA2
mininet@mininet-vm:~/PA2$ python TCPServer.py
The server is waiting to receive 2 connections....

Accepted first connection, calling it client X
Accepted second connection, calling it client Y
Waiting to receive messages from client X and client Y....
```

Case 1: When you type a message from Client X first:

Note: Client X connected before Client Y to the server.



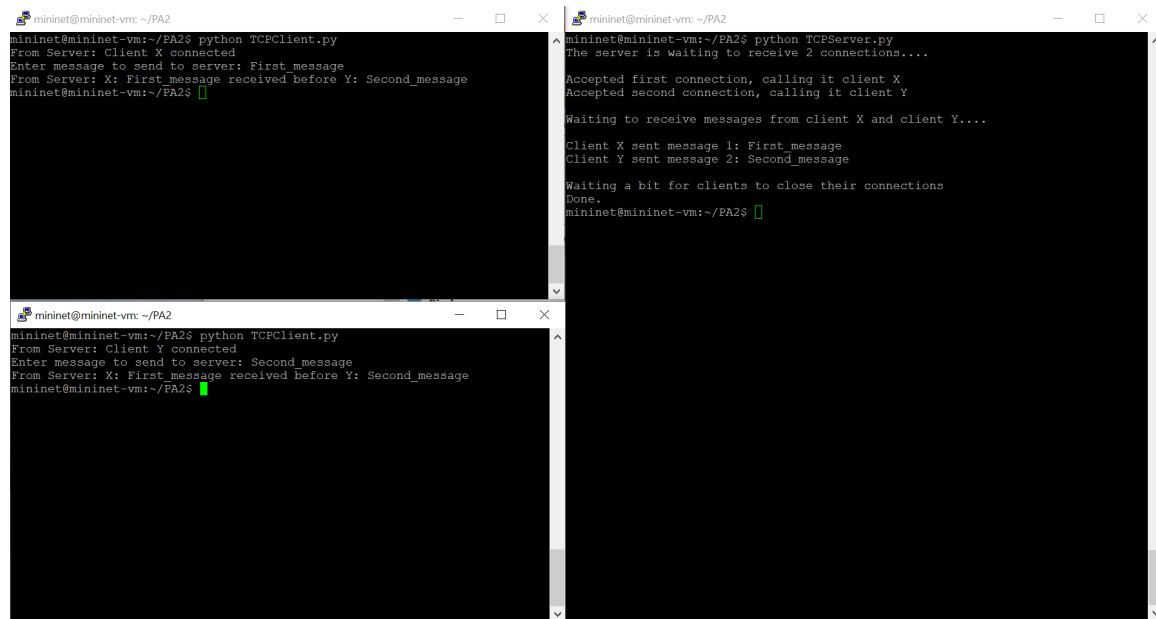
```
mininet@mininet-vm: ~/PA2
mininet@mininet-vm:~/PA2$ python TCPCClient.py
From Server: Client X connected
Enter message to send to server: First_message

mininet@mininet-vm: ~/PA2
mininet@mininet-vm:~/PA2$ python TCPCClient.py
From Server: Client Y connected
Enter message to send to server:

mininet@mininet-vm: ~/PA2
mininet@mininet-vm:~/PA2$ python TCPServer.py
The server is waiting to receive 2 connections....

Accepted first connection, calling it client X
Accepted second connection, calling it client Y
Waiting to receive messages from client X and client Y....

Client X sent message 1: First_message
```



```
mininet@mininet-vm: ~/PA2$ python TCPCClient.py
From Server: Client X connected
Enter message to send to server: First_message
From Server: X: First_message received before Y: Second_message
mininet@mininet-vm: ~/PA2$

mininet@mininet-vm: ~/PA2$ python TCPServer.py
The server is waiting to receive 2 connections....

Accepted first connection, calling it client X
Accepted second connection, calling it client Y

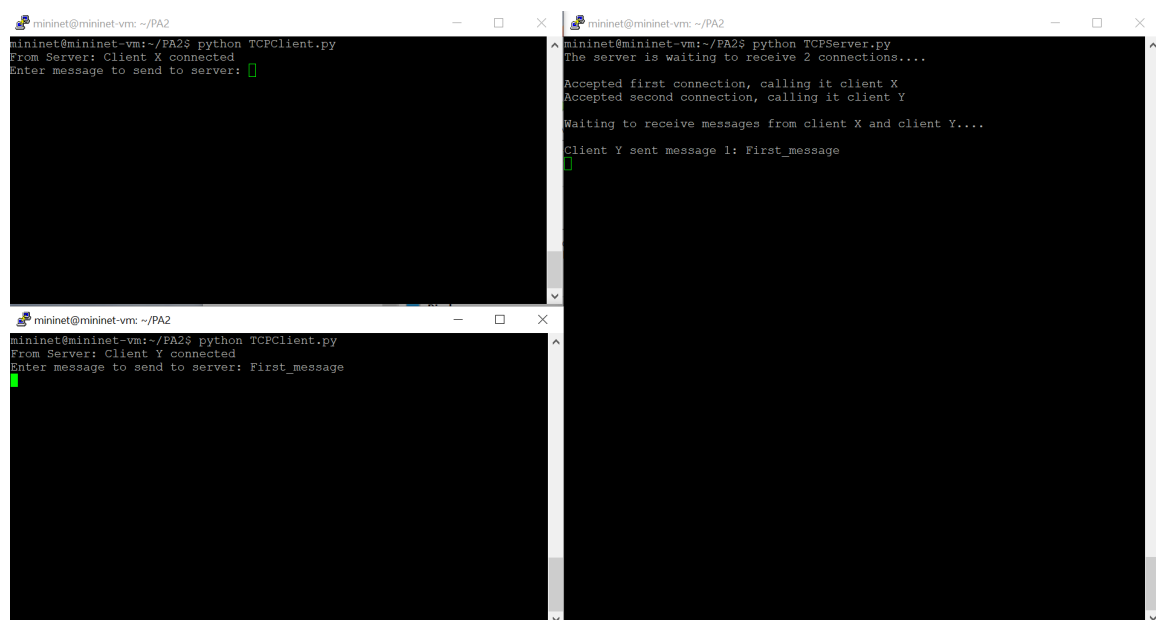
Waiting to receive messages from client X and client Y....

Client X sent message 1: First_message
Client Y sent message 2: Second_message

Waiting a bit for clients to close their connections
Done.
mininet@mininet-vm: ~/PA2$
```

Case 2: When you type a message from Client Y first:

Note: Client X connected before Client Y to the server.



```
mininet@mininet-vm: ~/PA2$ python TCPCClient.py
From Server: Client X connected
Enter message to send to server:

mininet@mininet-vm: ~/PA2$ python TCPServer.py
The server is waiting to receive 2 connections....

Accepted first connection, calling it client X
Accepted second connection, calling it client Y

Waiting to receive messages from client X and client Y....

Client Y sent message 1: First_message

mininet@mininet-vm: ~/PA2$ python TCPCClient.py
From Server: Client Y connected
Enter message to send to server: First_message
mininet@mininet-vm: ~/PA2$
```

```
mininet@mininet-vm: ~/PA2
mininet@mininet-vm:~/PA2$ python TCPClient.py
From Server: Client X connected
Enter message to send to server: Second_message
From Server: Y: First_message received before X: Second_message
mininet@mininet-vm:~/PA2$

mininet@mininet-vm: ~/PA2
mininet@mininet-vm:~/PA2$ python TCPServer.py
The server is waiting to receive 2 connections....
Accepted first connection, calling it client X
Accepted second connection, calling it client Y
Waiting to receive messages from client X and client Y....
Client Y sent message 1: First_message
Client X sent message 2: Second_message
Waiting a bit for clients to close their connections
Done.
mininet@mininet-vm:~/PA2$

mininet@mininet-vm: ~/PA2
mininet@mininet-vm:~/PA2$ python TCPClient.py
From Server: Client Y connected
Enter message to send to server: First_message
From Server: Y: First message received before X: Second_message
mininet@mininet-vm:~/PA2$
```

Message Format

The messages in this assignment are formatted in a simple way. The client message must look like:

Client X: Alice

Client Y: Bob

The messages from server to client must look like (depending on the order of the received messages):

X: Alice received before Y: Bob

OR

Y: Bob received before X: Alice

Grading Criteria

Your client and server programs need to demonstrate an understanding of the connection-oriented nature of the TCP protocol.

Specifically, your client program should:

1. **(5 points)** You must use TCP sockets; you will need to establish a connection first, since it is a connection oriented protocol.

2. **(5 points)** Clients must initiate the connection by sending their connection requests to the server one client at a time.

Specifically, your server program should:

3. **(5 points)** The server must accept connections from both clients first **BEFORE** receiving the messages from either client.
4. **(5 points)** Server establishes the first client that made a connection as Client X and the second one as client Y.
5. **(5 points)** After establishing a connection with both Clients, Server sends a message to both clients stating that a connection has been established. The message from the server must indicate which client is X and which client is Y. (Message to clients must look like: "Client X connected" or "Client Y connected".)
6. **(5 points)** Next, the server receives messages from both clients (in any order) and establishes which message it received first.
7. **(10 points)** Server sends acknowledgements to both clients stating which message was received first. (Message to clients must look like: "Y: First_message received before X: Second_message" or "X: First_message received before Y: Second_message".)
8. **(10 points)** The response string from Server to Clients ("X: First_message received before Y: Second_message", or "Y: First_message received before X: Second_message") must be in the order the messages from Client X or Client Y are received at the server and **NOT** the order in which the clients X and Y connected to the server. **Note: You will need multithreading and a way to share data between threads to achieve this.**
9. **(10 points)** Your program should print out the messages received by the client and server at the receiving end.
10. **(10 points)** Execute your programs on your mininet virtual machine.
11. **(20 points)** Explain why you need multithreading to solve this problem. Put this in a comment at the top of your server code.
12. **(10 points)** Program must be readable and well documented. All files must be in order as indicated in the "What to turn in" section.
13. Teamwork grade: **(50 points)** Each team member will grade each other teammate out of 10 points during peer evaluation. I will average all team members' grades and scale it to get your teamwork grade out of **50 points**. Note that 30% of your grade will come from your teamwork and team member evaluations.

What to Hand in

1. You will hand in the complete client and server code to iLearn. You only need one client program.
2. Minutes of the 3 meetings.
3. Make a pdf file with screenshots of server and client side output as shown in the expected output section. **Please do not upload the screenshots as image files.**

- a. Screenshots showing each client making a connection with the server separately (2 screenshots expected here).
- b. Screenshots showing Client X and Client Y (as established by the server) sending messages to the server separately in different order that is:
 - i. Client X sends a message to the server first then Client Y sends a message to the server. (2 screenshots expected here).
 - ii. Client Y sends a message to the server first then Client X sends a message to the server. (2 screenshots expected here).

Optional Extra-credit Exercises

The extra credit part of this assignment is mainly for fun and is challenging. You can modify the above server-client to create a simple chat service.

1. (10 points) Clients X and Y can only chat through the server. For example, every message that client X sends to the server, the server relays to client Y and vice versa.
2. (5 points) When a client (say X) wants to exit the chat service it sends a “Bye” message. When a server sees a “Bye” message, it relays this message to Y and then terminates the connection to both clients.
3. (5 points) Each client (say X) should output the messages sent by it and those received from Y. As this is a chat service the number/content of messages exchanged is not fixed. So, your clients should have the capability to accept inputs (which are the content of the messages) from the keyboard.

Example TCP Client-Server Code

The following code fully implements a capitalization server and client.

Server Code

```
#TCPCapitalizationServer.py
```

```
from socket import *
```

```
serverPort = 12000
```

```
# Create a TCP socket
```

```
# Notice the use of SOCK_STREAM for TCP
```

```
packets serverSocket =
```

```
socket(AF_INET,SOCK_STREAM) # Assign IP
```

```
address and port number to socket
```

```
serverSocket.bind('',serverPort))

serverSocket.listen(1) print ('The server is
ready to receive')

while True:

    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()

    capitalizedSentence = sentence.upper()

    connectionSocket.send(capitalizedSentence.encode())

    connectionSocket.close()
```

Client Code

```
from socket import *

# In your command prompt, type in hostname and press enter.

# What comes up is your computer's hostname

serverName = 'put your hostname here'

serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)

clientSocket.connect((serverName,serverPort))

sentence = input('Input lowercase sentence:')

clientSocket.send(sentence.encode())

modifiedSentence = clientSocket.recv(1024)

print ('From Server:', modifiedSentence.decode())

clientSocket.close()
```