**Networking -­ M1 international-­ Lab: Transport**

**Cyril Naves Samuel & Gabriel Pires**

# Socket TCP

In this laboratory, we are going to explore the basis of socket programming by using python3. You could find tutorials, documentation and examples in the next websites: https://docs.python.org/2/howto/sockets.html and https://docs.python.org/2/library/socket.html.

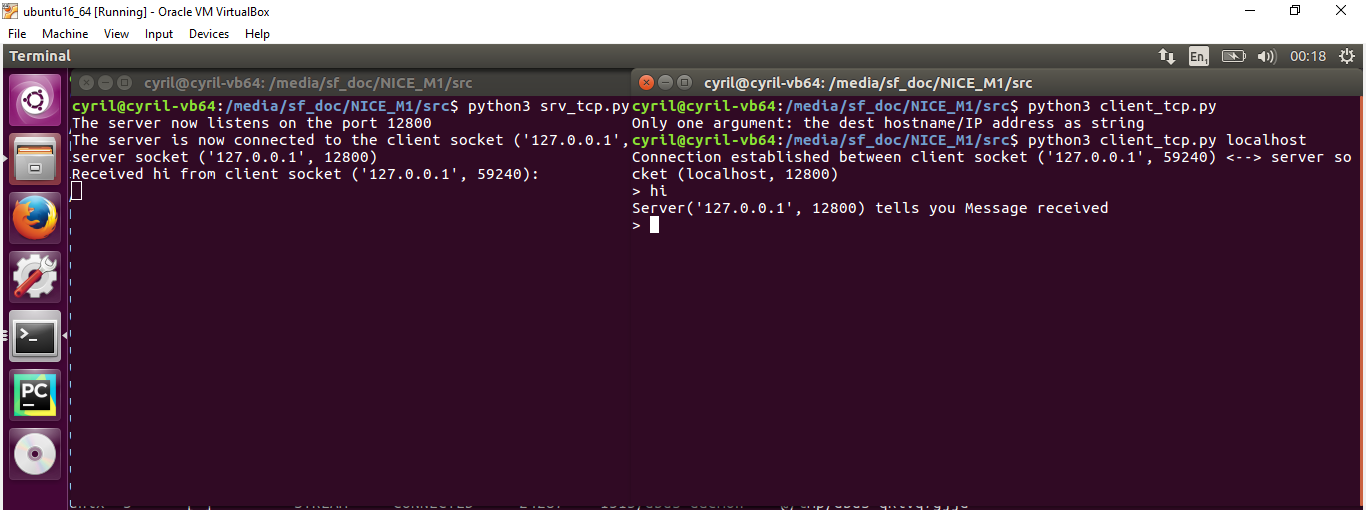
Download from my site the python3 files: client\_tcp.py and srv\_tcp.py. As suggested by the names, they implement a TCP client and a TCP server. The client needs as argument an IP address or a valid domain name, for example, ‘127.0.0.1’ or ‘localhost,’ if you want to use your own machine.

1. Using the command prompt, run $ python3 srv\_tcp.py and $ python3 client\_tcp.py <server\_name/server\_address>. Check that both processes communicate each other.

**Ans:**

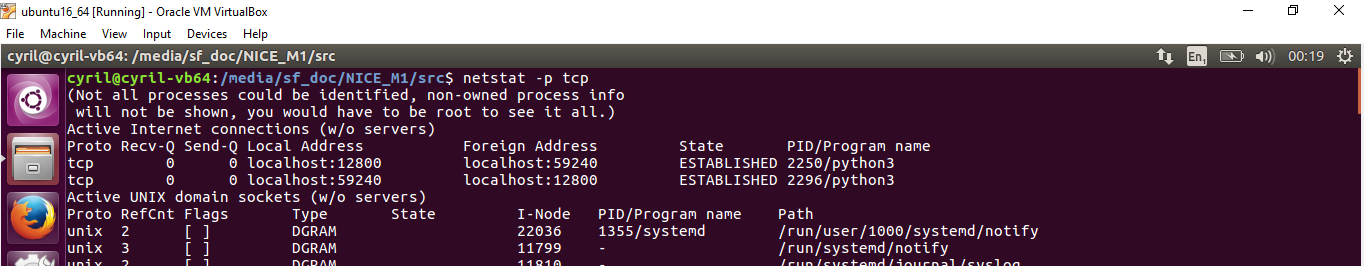
**Here Server and Client Connection is established on local host with Port 12800**

**Message String Sent & Received:1) hi**



1. Use the UNIX command netstat -p tcp *(see https://linux.die.net/man/8/netstat)* in order to visualize the active network connections (sockets) used by the protocols (tcp / udp). Normally there will be 7 columns. Explains the meaning of each for the connections you just opened. If too many connections are shown, you can also use the command lsof -n -i:12800 (*see* *https://linux.die.net/man/8/lsof*) to check the processus listening on a given port (in that case the port 12800). Which are the PID (process identifier) of the client and server processes? Why are there two lines for the server process?

**Ans:**



**Active Internet connections (w/o servers)**

**Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name**

**tcp 0 0 localhost:12800 localhost:59240 ESTABLISHED 2250/python3**

**tcp 0 0 localhost:59240 localhost:12800 ESTABLISHED 2296/python3**

**Proto 🡪 Refers to Socket used protocol used for communication, here it is TCP**

**Recv-Q🡪 Data in Queue for receiving, which is 0 so no delay in data to be read**

**Send-Q🡪 Data in Queue to sent ,which is also 0 so no delay**

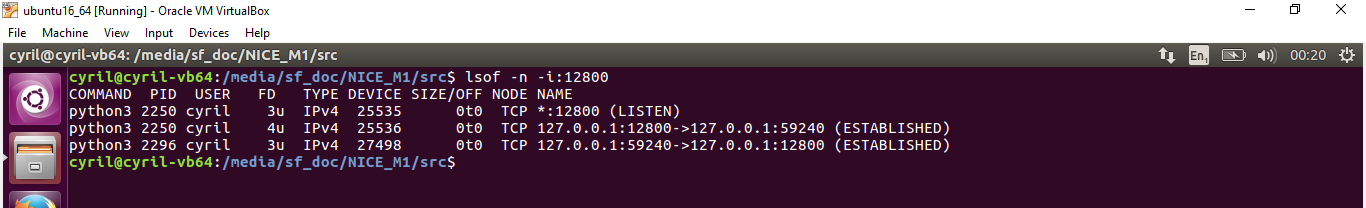
**Local Address🡪Address( IP) and Port from which the client or server originates**

**Foreign Address🡪 Address( IP) and Port from which the client or server connects to or destination**

**State – Specifies the state of the TCP connection which is in this case ESTBLISHED**

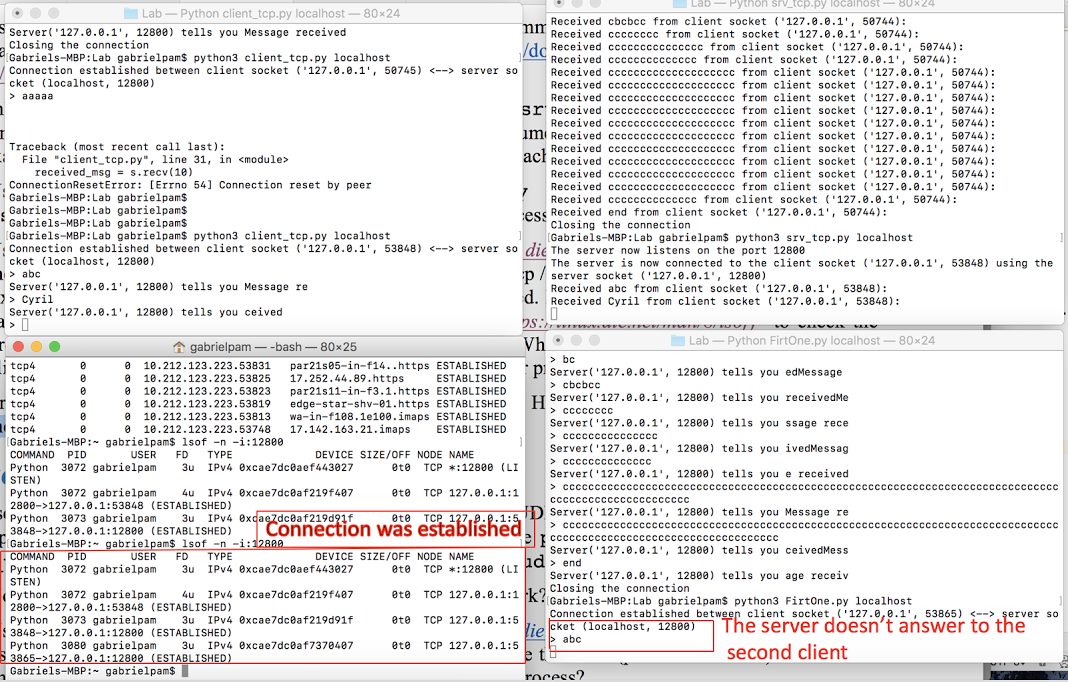
**PID/Program name – Process ID involved with the communication**

**Client PID**: 2296 **Server PID**: 2250



There are two lines for the server process because:

1. PID: 2250 first line is for the process to listening state of the server
2. PID: 2250 Second Line is for the process to communicate to the client which is in established state
3. Try to connect a second client to the server. Does it work? Why? Hint: the key is the blocking nature of the method recv. **Ans:**



By default, TCP sockets are in "blocking" mode. For recv() method to read from a stream, control isn't returned from program until at least one byte of data is read from the remote site. This process of waiting for data to appear is referred to as "blocking".

In second client connection is established but the message is not received or sent because while creating the object for Socket -🡪 **s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)** it is set to the default of blocking mode for recv() which has to be altered via **socket.setBlocking(flag) via 0 🡪 non-blocking and 1🡪 blocking.**

# Socket UDP

One solution to the problem detected in the last question is to use UDP instead of TCP. Using as template the example seen in the slides about “socket programming,” modify the python3 files: client\_tcp.py and srv\_tcp.py to use UDP sockets. Name the new files as client\_udp.py and srv\_udp.py.

**Source Code:**

**client\_udp.py**

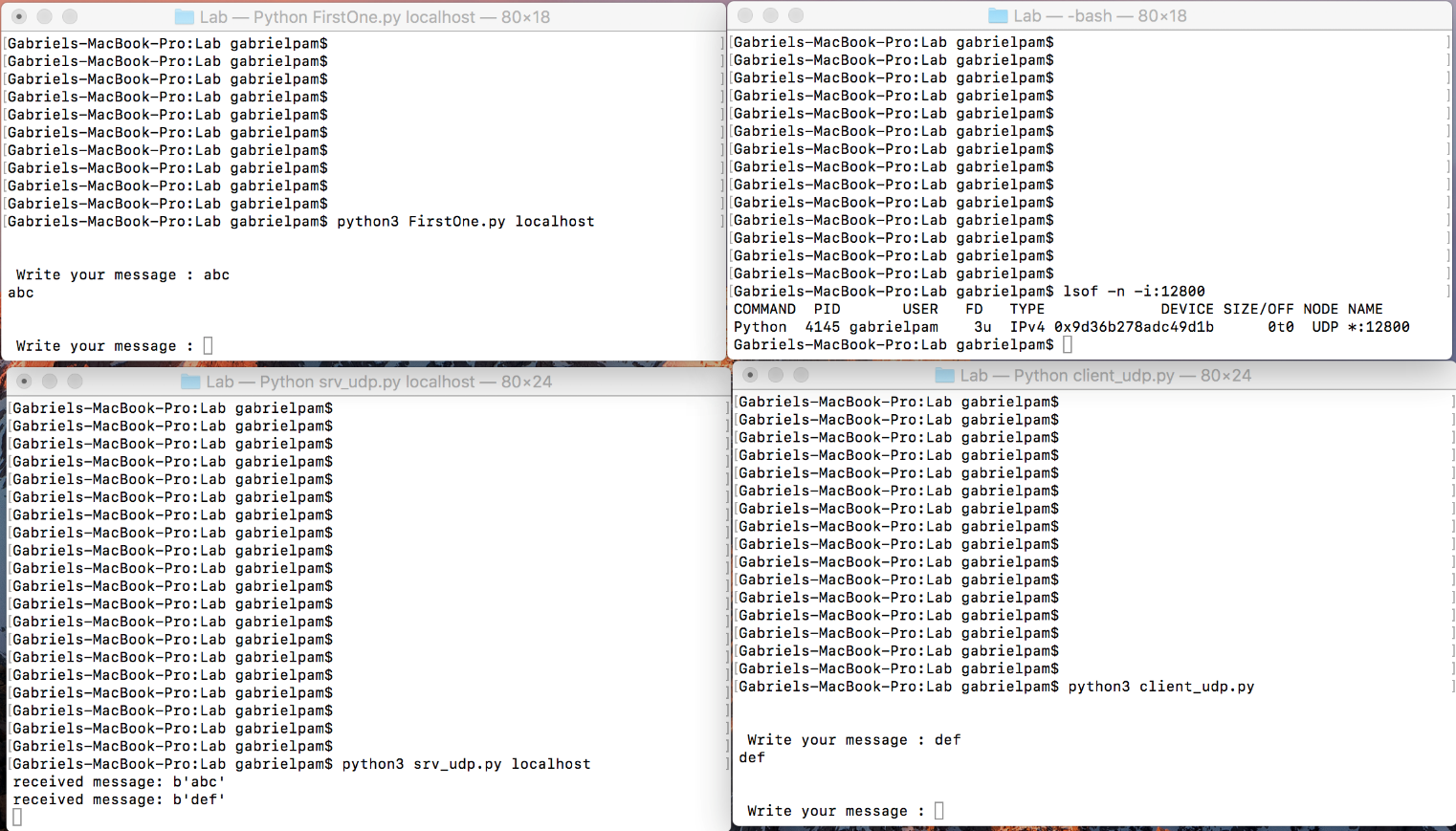
**import** socket  
**import** sys  
**import** struct  
  
**if** len(sys.argv) != 2:  
 print(**"Only one argument: the dest hostname/IP address as string"**)  
 sys.exit(2)  
  
*#Remote address*localhost = str(sys.argv[1]) *# dest hostname  
#host = "localhost"*localport = 12800  
  
s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
local\_addr = s.getsockname()  
print(**"UDP Socket established between client socket {0} <--> server socket ({1}, {2})"**.format(local\_addr, localhost, localport))  
  
msg\_to\_send = **b""  
while** msg\_to\_send != **b"end"**:  
 msg\_to\_send = input(**"> "**)  
 msg\_to\_send = bytes(msg\_to\_send,**'utf8'**)  
 *# We send the message* s.sendto(msg\_to\_send,(localhost,localport))  
 (received\_msg,(HOST,PORT)) = s.recvfrom(1024)  
 print(**"Server{0} tells you {1}"**.format(HOST, received\_msg))  
print(**"Closing the UDP Socket"**)  
s.close()

**srv\_udp.py:**

**import** socket  
**import** struct  
  
localhost = **''**port = 12800  
  
*# s is the "UDP" socket object*s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
s.bind((localhost, port))  
print(**"The server now bound on the port {}"**.format(port))  
  
data = **b""  
while** data != **b"end"**:  
 data, (HOST, PORT) = s.recvfrom(50)  
 s.sendto(**b"Message received"**,(HOST,PORT))  
 *# The following statement can throw an exception if the   
 # received message has special* print(**"Received {0} from client socket {1}: "**.format( data, HOST))  
   
print(**"Closing the UDP socket"**)  
s.close()

1. Now, try again to connect several clients to the server. Does it work? Why?

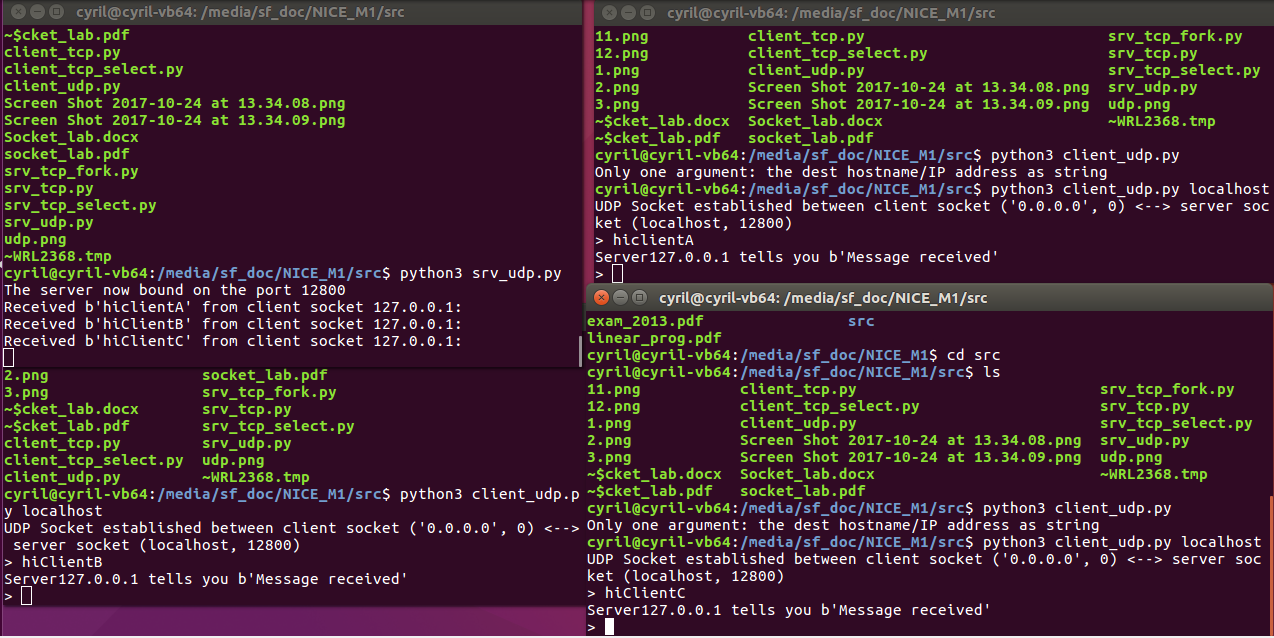
**Ans:**



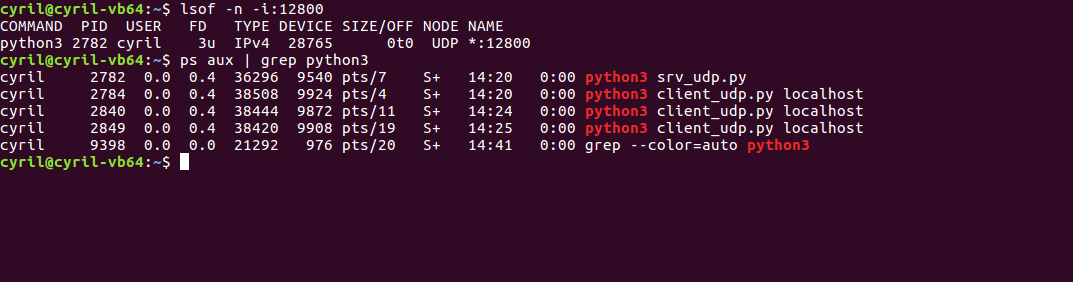
Sever accepts connection from several clients since it uses UDP protocol and it’s a connectionless one which can server multiple requests

1. Use the command lsof -n -i:12800 (*see* *https://linux.die.net/man/8/lsof*) to check the processes listening on a given port (in that case the port 12800). Which are the PIDs (process identifier) of the client and server processes? Why are there only one line for the server process?

**Ans:**



**Processes Window: lsof -n -i:12800 && ps aux| grep python3**



**Server Process: lsof -n -i:12800**

PID: 2782 : There is only one line for the server process since it is UDP and there are no separate states as **Listening** and **Established** like in TCP, UDP server just sends the message and forgets like “fire and forget” ince it is not reliable protocol like TCP

**Client Process: ps aux| grep python3**

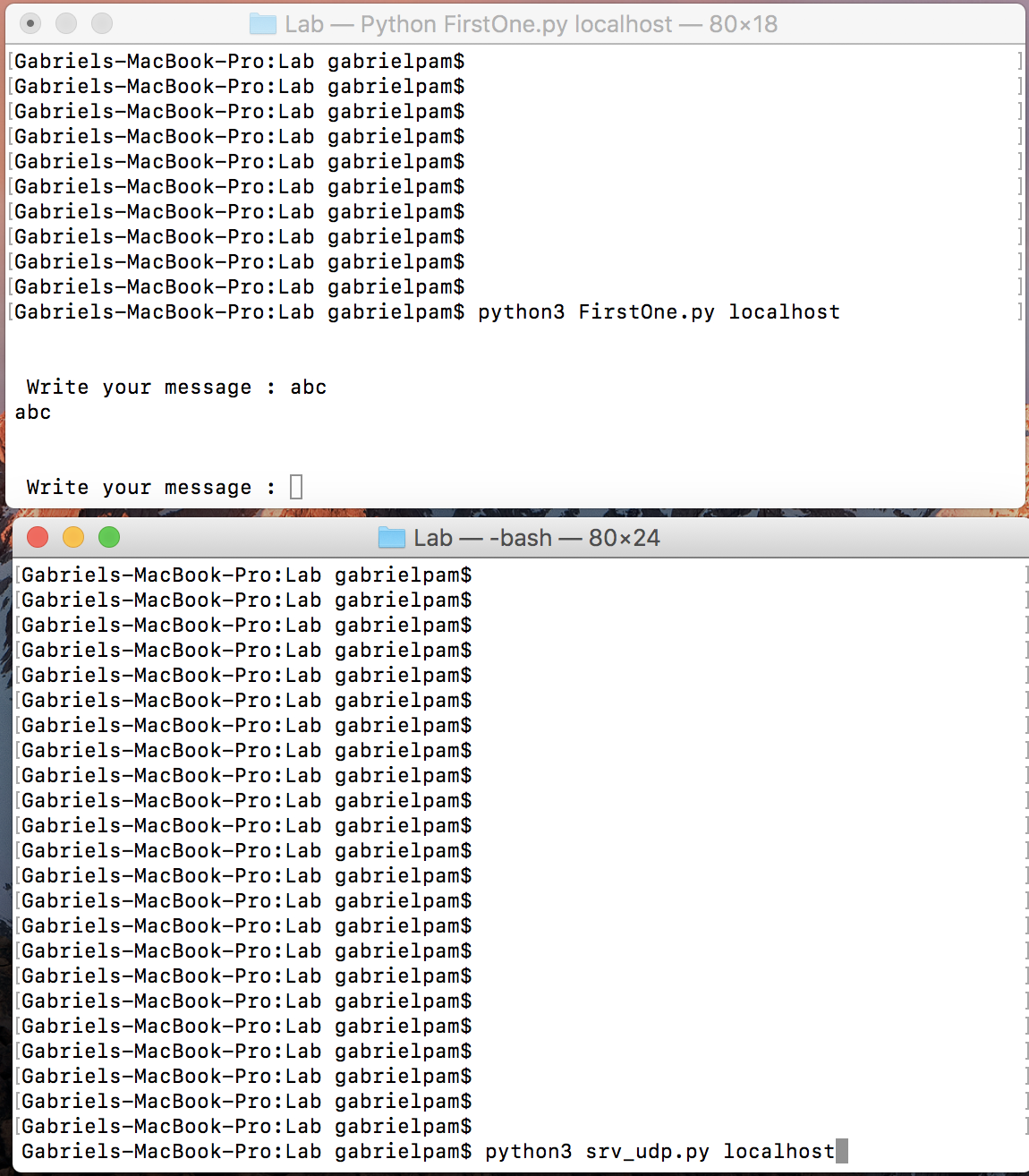
PID: 2784 First Client -Process

PID:2840 Second Client -Process

PID:2849 Third Client Process

1. Investigate and explain what happen if
   1. You start the UDP client before the server.

**Ans:**

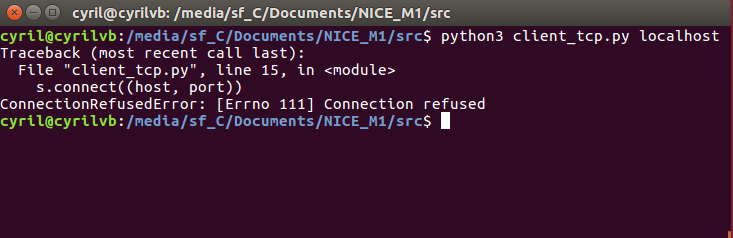


Here client side of UDP is executed but it does not transmit the message because there is no active server which can receive the data.

Since it does not establish a connection with the destination server for UDP,so the client is able to execute but unable to transmit data.

* 1. You start the TCP client before the server.

**Ans:**

Why?

**Ans:** In TCP Client tries to establish a connecting using ***socket.connect(host,port)*** where it expects a destination host server to establish a connection and go to listen state and then established state for a connection.

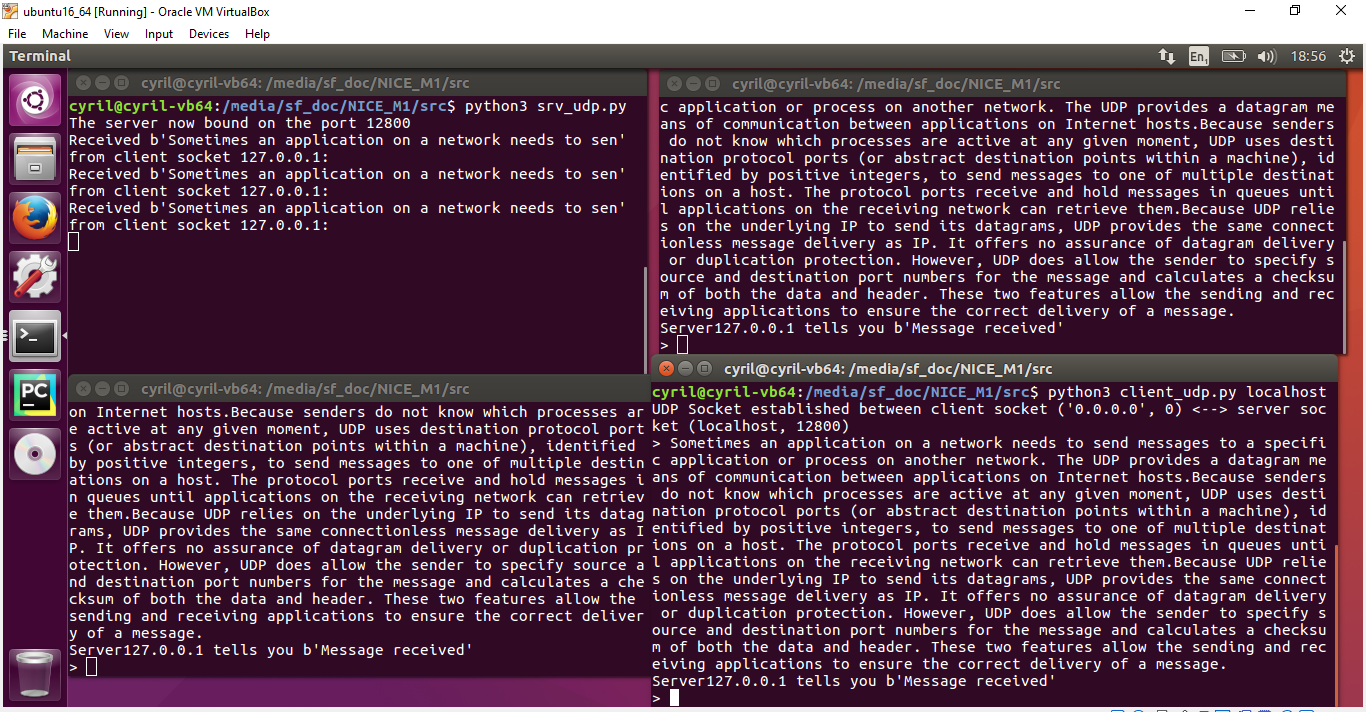
Since it does not find an active listening server for connection establishment, it is refused since it is a connection oriented protocol

1. Using many client processes from several machines, type and send long texts to force packet losses in the UDP case. (Note:)

Packet Message To be communicated between client and server:

***Sometimes an application on a network needs to send messages to a specific application or process on another network. The UDP provides a datagram means of communication between applications on Internet hosts.Because senders do not know which processes are active at any given moment, UDP uses destination protocol ports (or abstract destination points within a machine), identified by positive integers, to send messages to one of multiple destinations on a host. The protocol ports receive and hold messages in queues until applications on the receiving network can retrieve them.Because UDP relies on the underlying IP to send its datagrams, UDP provides the same connectionless message delivery as IP. It offers no assurance of datagram delivery or duplication protection. However, UDP does allow the sender to specify source and destination port numbers for the message and calculates a checksum of both the data and header. These two features allow the sending and receiving applications to ensure the correct delivery of a message.***

***Ans:***



Here Packet Loss happens in the original text being sent received only **“Sometimes an application on a network needs to sen”**, other text message is being forced to lose.

# Multi server TCP

Now, we come back to TCP. Try to transform the original srv\_tcp.py code into a multi TCP server, that is, a server able to accept several clients. To do that, you can use basically two styles: a forking (multithread) server or a concurrent single server (based on method select). The first style forks from the listening parent process as many child sub processes as incoming client connections. The second style uses select to simultaneously wait over several client sockets. When one of the clients, send something, the server wakes up. About the forking (multithread) serve, you have information in the slides. About the select style, you can also use info from site as https://pymotw.com/2/select/.

Once done, repeat the same questions:

**Ans:**

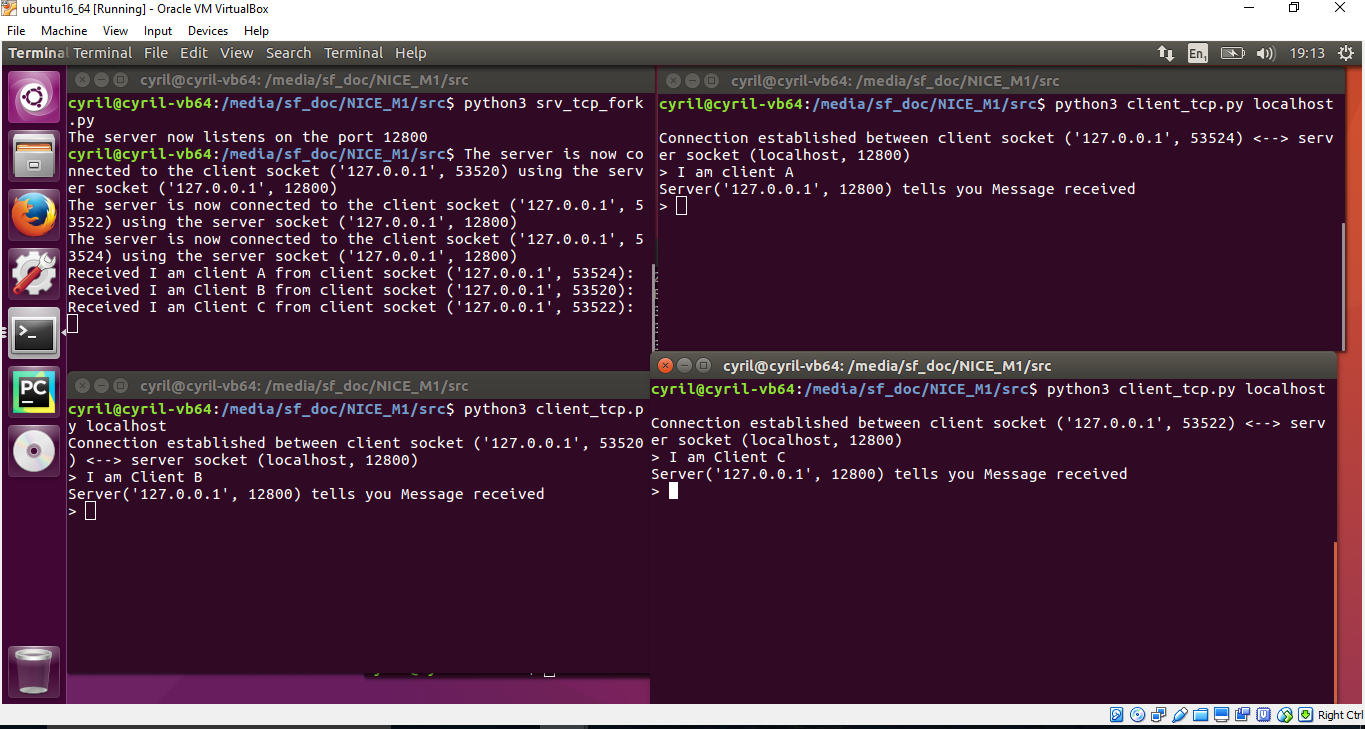
1. **Code Execution using Forking ( MultiThreading)**

**srv\_tcp\_fork.py**

**import** socket  
**import** os  
  
host = **''**port = 12800  
  
*# s is the "listening" socket object*s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
s.bind((host, port))  
s.listen(1)  
print(**"The server now listens on the port {}"**.format(port))  
  
  
**def** receive(s ,conn):  
 received\_msg = **b""  
 while** received\_msg != **b"end"**:  
 received\_msg = conn.recv(1024)  
 conn.send(**b"Message received"**)  
 *# The following statement can throw an exception if the  
 # received message has special* print(**"Received {0} from client socket {1}: "**.format(received\_msg.decode(), remote\_addr))  
 print(**"Closing the connection"**)  
 conn.close()  
 s.close()  
 **return** 1  
  
**for** i **in** range(0,5):  
 child\_pid = os.fork()  
 **if** child\_pid==0:  
 conn, remote\_addr = s.accept()  
 local\_addr = conn.getsockname()  
 print(**"The server is now connected to the client socket {0} using the server socket {1}"**.format(remote\_addr,local\_addr))  
 a = receive(s,conn)  
 **if**(a==1):  
 **break  
 else**:  
 i=+1

***For FORK (MultiThreading) Method:***

1. Try again to connect several clients to the server. Does it work? Why?

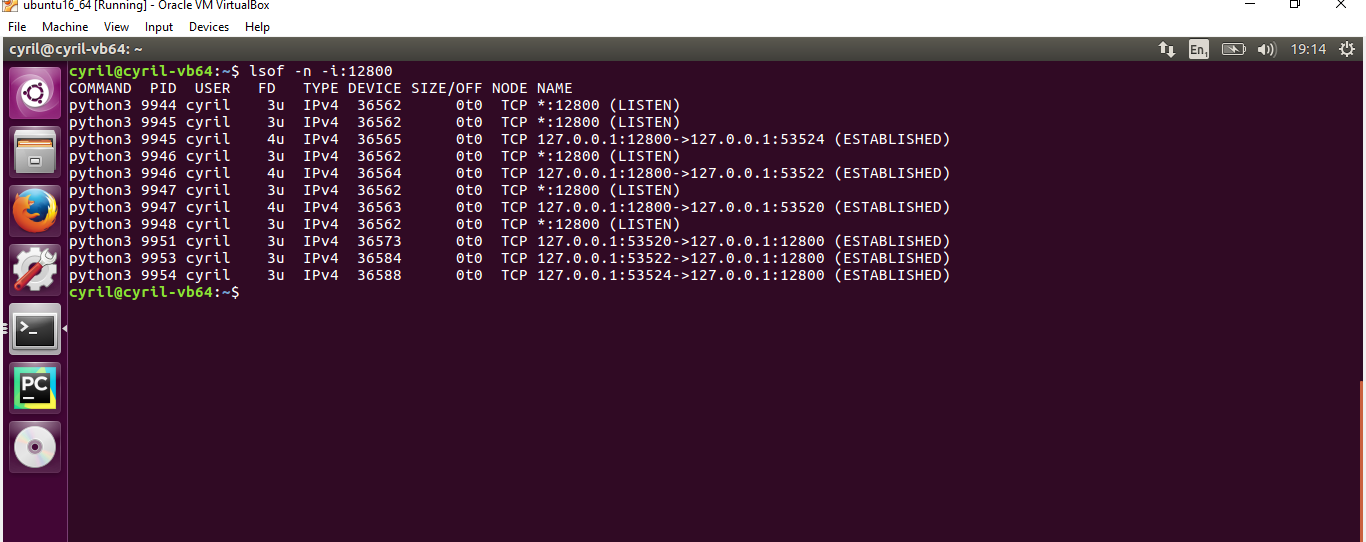


In this fork implementation, several server process are spawned from the parent process and each server process accepts incoming connection from the clients even though it is a TCP connection but with multiple forked processes.

1. Use the command lsof -n -i:12800 (*see* *https://linux.die.net/man/8/lsof*) to check the processes listening on a

given port (in that case the port 12800). Which are the PIDs (process identifier) of the client and server processes? Why are there several lines for the server process?

**Ans:**



**Here PID’s of Server:**

1) 9944

2) 9945

3) 9946

4) 9947

5) 9948

**PID’s of Client:**

1)9951

2)9953

3)9954

There are several lines for the server process with PID: 9944,9945,9946,9947,9948 because a single server process has been forked in which each of them accepts incoming connection for multiple clients. Since the server is always in listening state, it has other lines for the ESTABLISHED state for the respective clients.

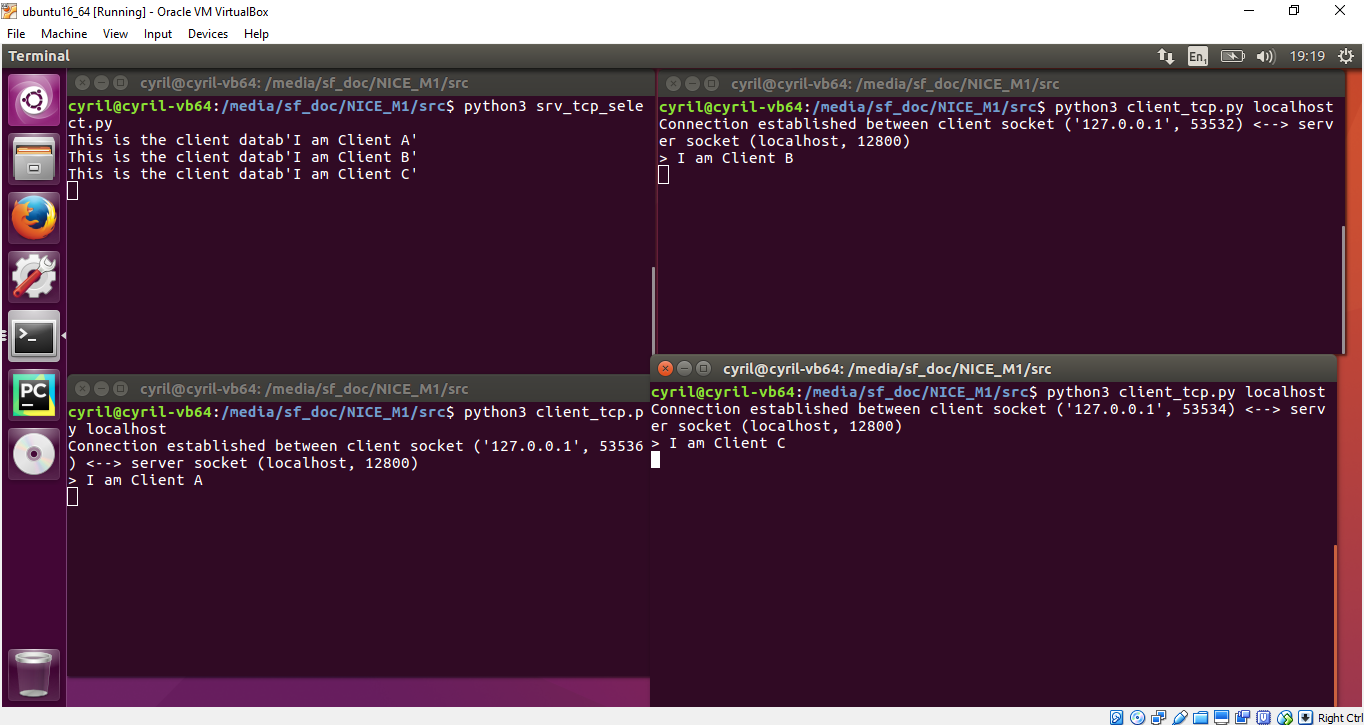
1. **Code Execution using Select:**

**srv\_tcp\_select.py**

**import** socket  
**import** select  
  
host=**''**port=12800  
  
socket=socket.socket(socket.AF\_INET , socket.SOCK\_STREAM)  
socket.bind((host,port))  
socket.listen(1)  
  
*#creating input array to be passed to select function for incoming, outgoing and error data*inputarray = [socket]  
  
**while** 1:  
 inputs,outputs,errors = select.select(inputarray,inputarray,[],5)  
 **if** (len(inputs)!= 0):  
 i=0  
 **for** input **in** inputs:  
 **if**(i<len(inputs)):  
 **if** input **is** socket:  
 clientsocket,clientaddress = input.accept()  
 inputarray.append(clientsocket)  
 **else**:  
 data=input.recv(1024)  
  
 **if not** data:  
 inputarray.remove(input)  
 **else**:  
 print (**"This is the client data"**+str(data))  
 i=i+1

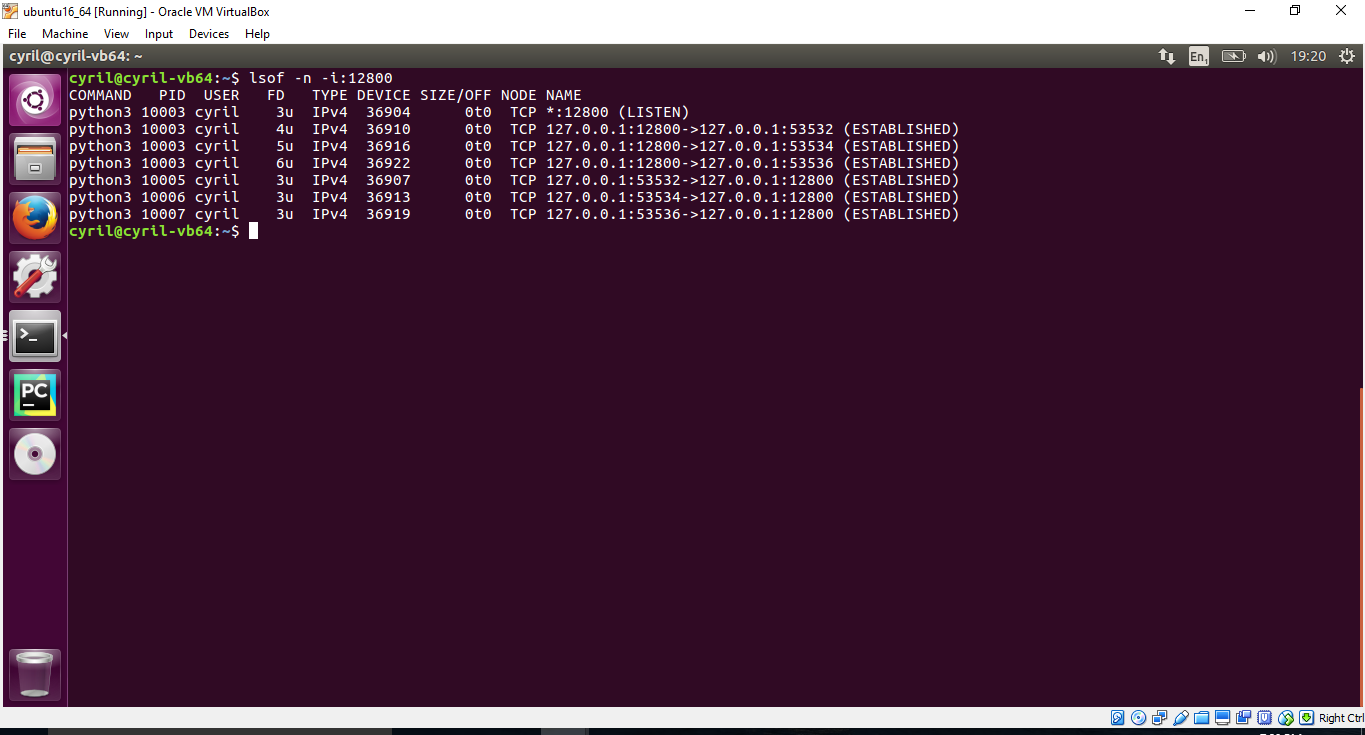
**For Select Method:**

1. Try again to connect several clients to the server. Does it work? Why?



Here in select () method which is an efficient method to wait for I/O ,which monitors sockets for incoming connections and as the multiple clients try to connect to the server, server accepts the incoming connection and also manages them simultaneously.

1. Use the command lsof -n -i:12800 (*see* *https://linux.die.net/man/8/lsof*) to check the processes listening on a given port (in that case the port 12800). Which are the PIDs (process identifier) of the client and server processes? Why are there several lines for the server process?



**Here Server PID’s:**

1. 10003

**Here Client PID’s:**

1. 10005
2. 10006
3. 10007

There are several lines for the server process with PID: 10003 because a single server using select method accepts incoming connection for multiple clients. Since the server is always in listening state, it has other lines for the ESTABLISHED state for the respective clients.