

Networking -- M1 international-- Lab: Transport

Cyril Naves Samuel & Gabriel Pires

1. 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>.

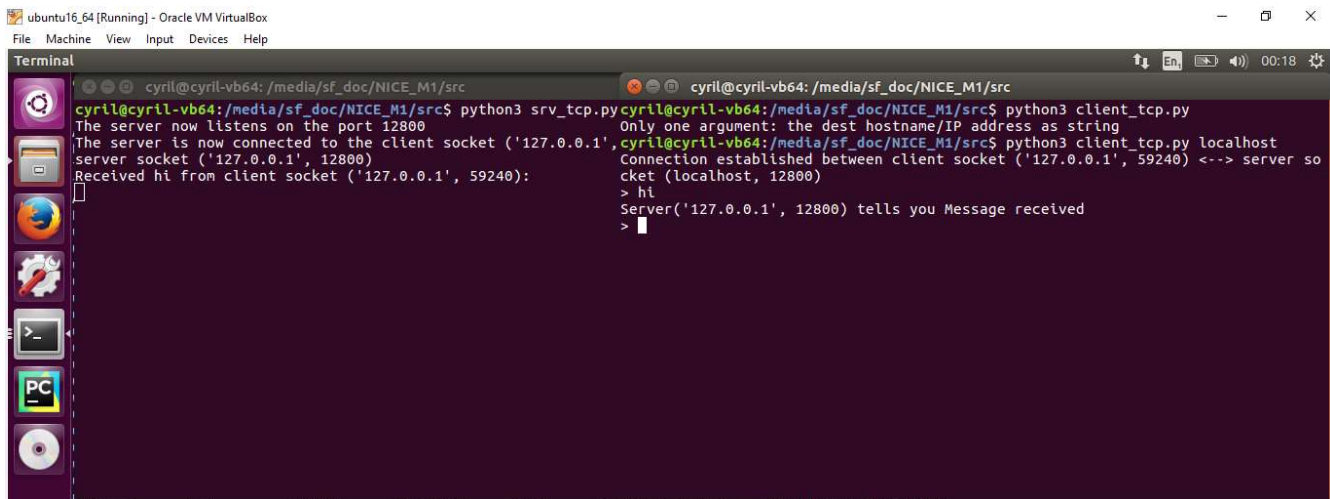
Download from my site the python3 files: `client_tcp.py` and `srv_tcp.py`. As suggested by the names, they implement a TCP client and a TCP server. The client needs as argument an IP address or a valid domain name, for example, '127.0.0.1' or 'localhost,' if you want to use your own machine.

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

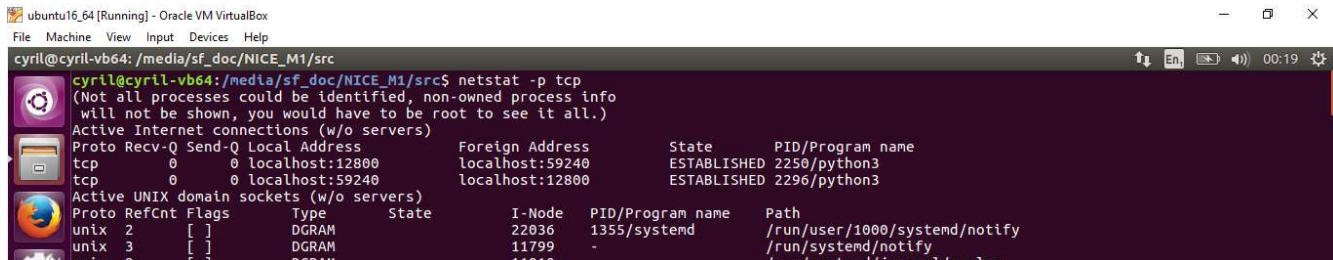


```
ubuntu16_64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal
cyril@cyril-vb64: /media/sf_doc/NICE_M1/src
cyril@cyril-vb64: /media/sf_doc/NICE_M1/src$ python3 srv_tcp.py
The server now listens on the port 12800
server socket ('127.0.0.1', 12800)
Received hi from client socket ('127.0.0.1', 59240):
cyril@cyril-vb64: /media/sf_doc/NICE_M1/src$ python3 client_tcp.py localhost
Only one argument: the dest hostname/IP address as string
Connection established between client socket ('127.0.0.1', 59240) <-> server socket (localhost, 12800)
> hi
Server('127.0.0.1', 12800) tells you Message received
>
```

2. 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 :



```
cyril@cyril-vb64: /media/sf_doc/NICE_M1/src$ netstat -p tcp
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
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
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags   Type       State      I-Node  PID/Program name  Path
unix    2      [ ]     DGRAM      -          22036   1355/systemd      /run/user/1000/systemd/notify
unix    3      [ ]     DGRAM      -          11799   -                 /run/systemd/notify
unix    2      [ ]     DGRAM      -          11810   -                 /run/systemd/journal/syslog
```

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

2. 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

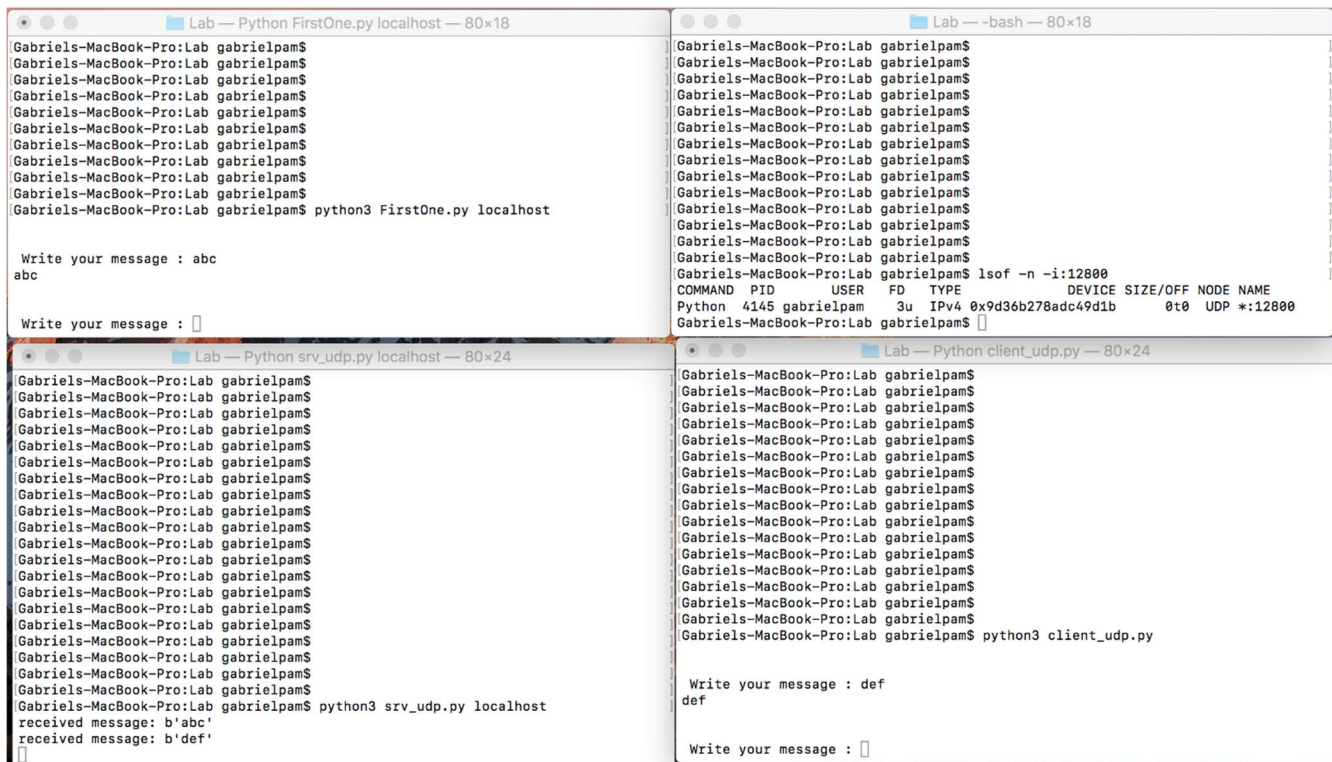
# s is the "UDP" socket object
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind((localhost, port))
print("The server now bound on the port {}".format(port))

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:



The image displays four terminal windows illustrating a UDP server-client interaction:

- Top Left (Python FirstOne.py localhost):** Shows the server script being executed. It prompts for a message, receives 'abc', and then waits for another input.
- Top Right (-bash):** Shows the system's process table using the `lsOF` command. It lists the `python3 FirstOne.py localhost` process with details like PID (4145), USER (gabrielpam), FD (3u), TYPE (IPv4), DEVICE (0x9d36b278adc49d1b), SIZE/OFF (0t0), and NODE NAME (UDP *:12800).
- Bottom Left (Python srv_udp.py localhost):** Shows the server script receiving multiple messages. It prints 'received message: b\'abc\'' and 'received message: b\'def\''.
- Bottom Right (Python client_udp.py):** Shows a client script being executed, which prompts for a message and then displays 'def'.

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

- 2) 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:

The first terminal window shows the execution of `python3 srv_udp.py`. The output indicates the server is bound on port 12800 and receives three messages from client socket 127.0.0.1: 'hiClientA', 'hiClientB', and 'hiClientC'.

The second terminal window shows the execution of `python3 client_udp.py localhost`. The output shows the client establishing a UDP socket and sending three messages to the server: 'hiClientA', 'hiClientB', and 'hiClientC'.

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

```
cyril@cyril-vb64:~$ lsof -n -i:12800
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
python3 2782 cyril 3u IPv4 28765 0t0 UDP *:12800

cyril@cyril-vb64:~$ ps aux | grep python3
cyril 2782 0.0 0.4 36296 9540 pts/7 S+ 14:20 0:00 python3 srv_udp.py
cyril 2784 0.0 0.4 38508 9924 pts/4 S+ 14:20 0:00 python3 client_udp.py localhost
cyril 2840 0.0 0.4 38444 9872 pts/11 S+ 14:24 0:00 python3 client_udp.py localhost
cyril 2849 0.0 0.4 38420 9908 pts/19 S+ 14:25 0:00 python3 client_udp.py localhost
cyril 9398 0.0 0.0 21292 976 pts/20 S+ 14:41 0:00 grep --color=auto 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” since it is not reliable protocol like TCP

Client Process: ps aux | grep python3

PID: 2784 First Client -Process

PID:2840 Second Client -Process

- 3) Investigate and explain what happen if
 - a. You start the UDP client before the server.

Ans:

The image shows two terminal windows. The top window, titled 'Lab — Python FirstOne.py localhost — 80x18', shows a series of 12 empty prompts for 'gabrielpam\$'. The 13th prompt is followed by the command 'python3 FirstOne.py localhost'. Below this, the prompt 'Write your message : ' is followed by the input 'abc'. The bottom window, titled 'Lab — -bash — 80x24', shows a series of 20 empty prompts for 'gabrielpam\$'. The 21st prompt is followed by the command 'python3 srv_udp.py localhost'.

```

Lab — Python FirstOne.py localhost — 80x18
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$ python3 FirstOne.py localhost

Write your message : abc
abc

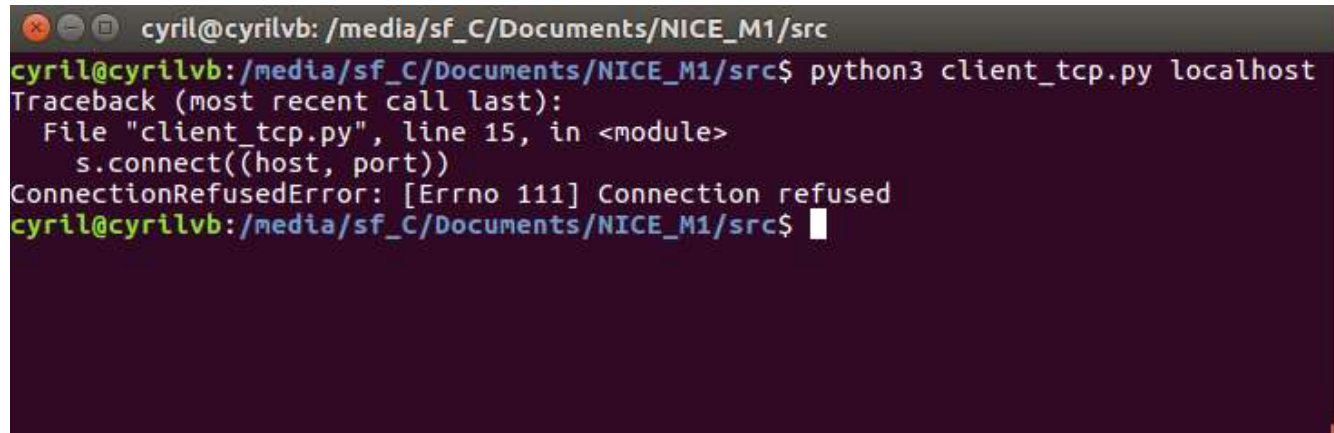
Write your message : 
Lab — -bash — 80x24
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$
Gabriels-MacBook-Pro:Lab gabrielpam$ python3 srv_udp.py localhost
  
```

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.

b. You start the TCP client before the server.

Ans:

A terminal window with a dark background. The prompt is 'cyril@cyrilvb: /media/sf_C/Documents/NICE_M1/src'. The user has entered 'python3 client_tcp.py localhost'. The output shows a traceback: 'Traceback (most recent call last):', 'File "client_tcp.py", line 15, in <module>', 's.connect((host, port))', and 'ConnectionRefusedError: [Errno 111] Connection refused'. The prompt returns to 'cyril@cyrilvb: /media/sf_C/Documents/NICE_M1/src\$'.

```
cyril@cyrilvb: /media/sf_C/Documents/NICE_M1/src
cyril@cyrilvb: /media/sf_C/Documents/NICE_M1/src$ python3 client_tcp.py localhost
Traceback (most recent call last):
  File "client_tcp.py", line 15, in <module>
    s.connect((host, port))
ConnectionRefusedError: [Errno 111] Connection refused
cyril@cyrilvb: /media/sf_C/Documents/NICE_M1/src$
```

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

- 4) 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:

```
Terminal
ubuntu16_64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

cyril@cyril-vb64: /media/sf_doc/NICE_M1/src
cyril@cyril-vb64:/media/sf_doc/NICE_M1/src$ python3 srv_udp.py
The server now bound on the port 12800
Received b'Sometimes an application on a network needs to sen'
from client socket 127.0.0.1:
Received b'Sometimes an application on a network needs to sen'
from client socket 127.0.0.1:
Received b'Sometimes an application on a network needs to sen'
from client socket 127.0.0.1:
>

cyril@cyril-vb64:/media/sf_doc/NICE_M1/src$ python3 client_udp.py localhost
UDP Socket established between client socket ('0.0.0.0', 0) <--> server soc
ket (localhost, 12800)
> Sometimes an application on a network needs to send messages to a specifi
c application or process on another network. The UDP provides a datagram me
ans of communication between applications on Internet hosts. Because senders
do not know which processes are active at any given moment, UDP uses desti
nation protocol ports (or abstract destination points within a machine), id
entified by positive integers, to send messages to one of multiple destinat
ions on a host. The protocol ports receive and hold messages in queues unti
l applications on the receiving network can retrieve them. Because UDP relie
s on the underlying IP to send its datagrams, UDP provides the same connect
ionless message delivery as IP. It offers no assurance of datagram delivery
or duplication protection. However, UDP does allow the sender to specify s
ource and destination port numbers for the message and calculates a checksu
m of both the data and header. These two features allow the sending and rec
eiving applications to ensure the correct delivery of a message.
Server127.0.0.1 tells you b'Message received'
>

cyril@cyril-vb64:/media/sf_doc/NICE_M1/src$
```

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.

3. 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

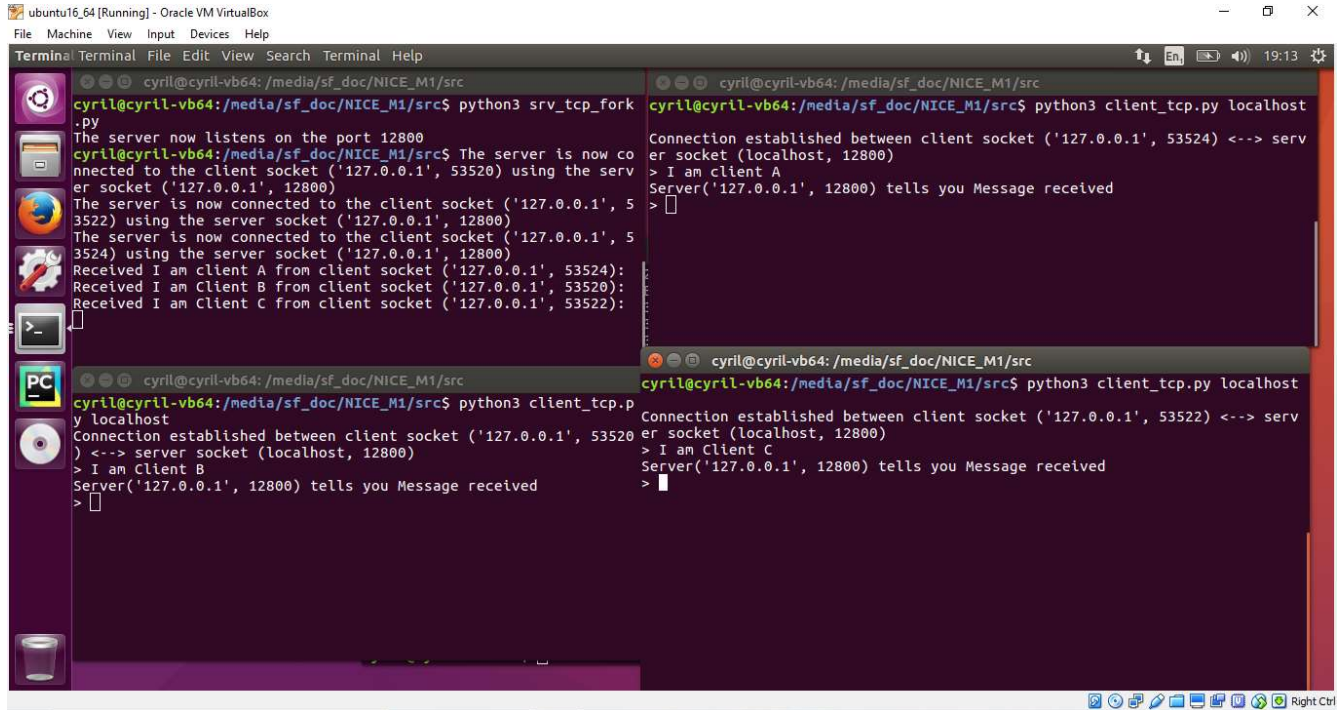
# s is the "listening" socket object
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((host, port))
s.listen(1)
print("The server now listens on the port {}".format(port))

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?



```
ubuntu16_64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal Terminal File Edit View Search Terminal Help

cyril@cyril-vb64: /media/sf_doc/NICE_M1/src
cyril@cyril-vb64:/media/sf_doc/NICE_M1/src$ python3 srv_tcp_fork.py
The server now listens on the port 12800
cyril@cyril-vb64:/media/sf_doc/NICE_M1/src$ The server is now connected to the client socket ('127.0.0.1', 53520) using the server socket ('127.0.0.1', 12800)
The server is now connected to the client socket ('127.0.0.1', 53522) using the server socket ('127.0.0.1', 12800)
The server is now connected to the client socket ('127.0.0.1', 53524) using the server socket ('127.0.0.1', 12800)
Received I am client A from client socket ('127.0.0.1', 53524):
Received I am Client B from client socket ('127.0.0.1', 53520):
Received I am Client C from client socket ('127.0.0.1', 53522):
>

cyril@cyril-vb64: /media/sf_doc/NICE_M1/src
cyril@cyril-vb64:/media/sf_doc/NICE_M1/src$ python3 client_tcp.py localhost
Connection established between client socket ('127.0.0.1', 53524) <--> server socket (localhost, 12800)
> I am client A
Server('127.0.0.1', 12800) tells you Message received
>

cyril@cyril-vb64: /media/sf_doc/NICE_M1/src
cyril@cyril-vb64:/media/sf_doc/NICE_M1/src$ python3 client_tcp.py localhost
Connection established between client socket ('127.0.0.1', 53520) <--> server socket (localhost, 12800)
> I am Client B
Server('127.0.0.1', 12800) tells you Message received
>

cyril@cyril-vb64: /media/sf_doc/NICE_M1/src
cyril@cyril-vb64:/media/sf_doc/NICE_M1/src$ python3 client_tcp.py localhost
Connection established between client socket ('127.0.0.1', 53522) <--> server socket (localhost, 12800)
> I am Client C
Server('127.0.0.1', 12800) tells you Message received
>
```

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.

2. 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:

```
cyril@cyril-vb64: ~$ lsof -n -i:12800
COMMAND PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
python3 9944 cyril   3u    IPv4  36562      0t0  TCP *:12800 (LISTEN)
python3 9945 cyril   3u    IPv4  36562      0t0  TCP *:12800 (LISTEN)
python3 9945 cyril   4u    IPv4  36565      0t0  TCP 127.0.0.1:12800->127.0.0.1:53524 (ESTABLISHED)
python3 9946 cyril   3u    IPv4  36562      0t0  TCP *:12800 (LISTEN)
python3 9946 cyril   4u    IPv4  36564      0t0  TCP 127.0.0.1:12800->127.0.0.1:53522 (ESTABLISHED)
python3 9947 cyril   3u    IPv4  36562      0t0  TCP *:12800 (LISTEN)
python3 9947 cyril   4u    IPv4  36563      0t0  TCP 127.0.0.1:12800->127.0.0.1:53520 (ESTABLISHED)
python3 9948 cyril   3u    IPv4  36562      0t0  TCP *:12800 (LISTEN)
python3 9951 cyril   3u    IPv4  36573      0t0  TCP 127.0.0.1:53520->127.0.0.1:12800 (ESTABLISHED)
python3 9953 cyril   3u    IPv4  36584      0t0  TCP 127.0.0.1:53522->127.0.0.1:12800 (ESTABLISHED)
python3 9954 cyril   3u    IPv4  36588      0t0  TCP 127.0.0.1:53524->127.0.0.1:12800 (ESTABLISHED)
cyril@cyril-vb64: ~$
```

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.

2) 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)

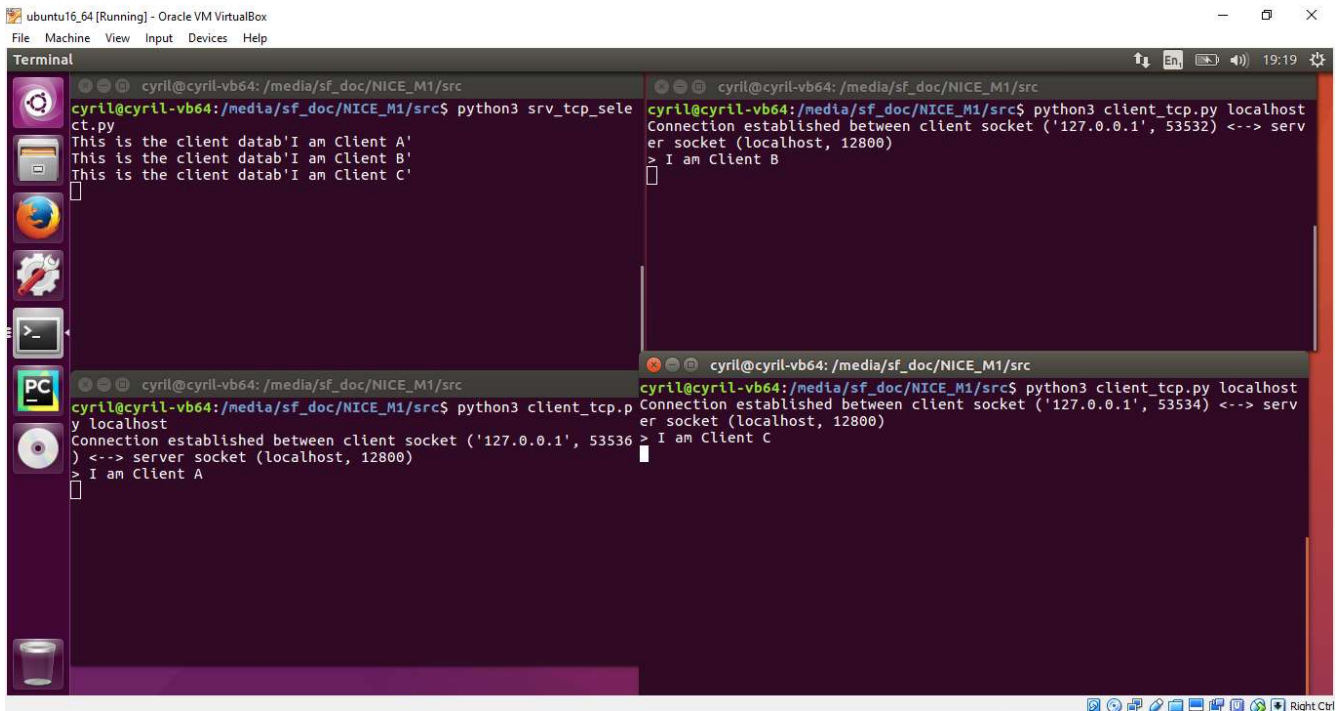
#creating input array to be passed to select function for incoming, outgoing and error data
inputarray = [socket]

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:

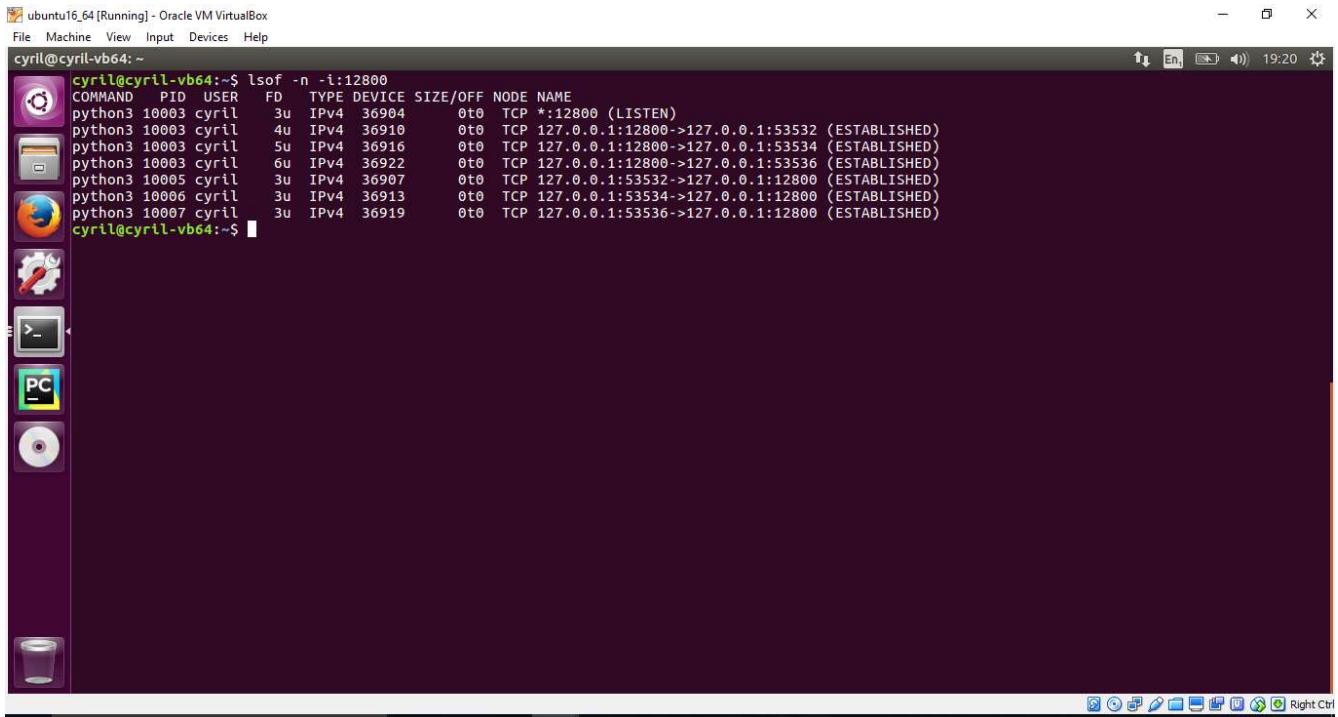
- a) Try again to connect several clients to the server. Does it work? Why?



The screenshot shows a VirtualBox window titled 'ubuntu16_64 [Running] - Oracle VM VirtualBox'. Inside, there are three terminal windows. The top-left terminal shows a server running `python3 srv_tcp_select.py` which prints 'This is the client datab'I am Client A'', 'This is the client datab'I am Client B'', and 'This is the client datab'I am Client C''. The top-right terminal shows a client running `python3 client_tcp.py localhost` which prints 'Connection established between client socket ('127.0.0.1', 53532) <--> server socket (localhost, 12800)' and 'I am Client B'. The bottom terminal shows another client running `python3 client_tcp.py localhost` which prints 'Connection established between client socket ('127.0.0.1', 53536) <--> server socket (localhost, 12800)' and 'I am Client A'.

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.

- b) 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?



```
cyril@cyril-vb64:~$ lsof -n -i:12800
COMMAND  PID  USER  FD  TYPE  DEVICE  SIZE/OFF  NODE  NAME
python3 10003 cyril  3u  IPv4  36904      0t0  TCP  *:12800 (LISTEN)
python3 10003 cyril  4u  IPv4  36910      0t0  TCP  127.0.0.1:12800->127.0.0.1:53532 (ESTABLISHED)
python3 10003 cyril  5u  IPv4  36916      0t0  TCP  127.0.0.1:12800->127.0.0.1:53534 (ESTABLISHED)
python3 10003 cyril  6u  IPv4  36922      0t0  TCP  127.0.0.1:12800->127.0.0.1:53536 (ESTABLISHED)
python3 10005 cyril  3u  IPv4  36907      0t0  TCP  127.0.0.1:53532->127.0.0.1:12800 (ESTABLISHED)
python3 10006 cyril  3u  IPv4  36913      0t0  TCP  127.0.0.1:53534->127.0.0.1:12800 (ESTABLISHED)
python3 10007 cyril  3u  IPv4  36919      0t0  TCP  127.0.0.1:53536->127.0.0.1:12800 (ESTABLISHED)
cyril@cyril-vb64:~$
```

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.