

CSLR 52

Networks Lab

Assignment 1

106119029

Dipesh Kafle

Questions

- 1) Write a server program for TCP using Python to do the following:
 - a. Server returns the binary value of the text sent by the client. Example: for a text string “commetsii”, the client should receive “01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001”
 - b. The server should be running at the localhost interface
 - c. You are free to choose any port
- 2) The Binary Decoding Server: Write another server running on eth1 interface to do the following:
 - a. Server returns the string value of a binary input. Example: the binary string “01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001” sent from the client should return “commetsii”
- 3) Write two client programs one for each of the server.
 - a. What happens if the client aborts (e.g., when the input is CTRL/D)?
 - b. Can you run two clients against the same server? Why or why not? Hint: Multithreaded socket server in Python
 - c. What happens when you try to connect client1 to server2 by passing a wrong network address (e.g., connecting to localhost instead of eth1 IP)?

Output (Single Threaded)

```
LabsAndAssignments/LabNetworks/Assignment1 on 3: cur_sem [x!?] via 2 v3.9.7
λ python server1.py
Interface: lo, Port: 3000, Multi-threaded?: False
Connected to ('127.0.0.1', 42506)
Received: comnetsii
Sent: 01000011 0101111 01101101 01101110 01100101 01110100 01110011 01101001 0110100
1
Received: comm
Sent: 01100011 01101111 01101101 01101110
Received: comnet
Sent: 01100011 01101111 01101101 01101110 01100101 01110100
Received: comnetsii
Sent: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 0110100
1
Received: comm
Sent: 01100011 01101111 01101101 01101110
Received: comnet
Sent: 01100011 01101111 01101101 01101110 01100101 01110100
Received: comnetsii
Sent: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 0110100
1
Received: abcdef
Sent: 01100001 01100010 01100011 01100100 01100101 01100110
LabsAndAssignments/LabNetworks/Assignment1 on 3: cur_sem [x!?] via 2 v3.9.7 took 20s
λ

LabsAndAssignments/LabNetworks/Assignment1 on 3: cur_sem [x!?] via 2 v3.9.7
λ python client1.py
Interface: lo, Port: 3000
('127.0.0.1', 3000)
comnetsii
Sent: comnetsii
Received: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001
1101001
comm
Sent: comm
Received: 01100011 01101111 01101101 01101110
comnet
Sent: comnet
Received: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 0110100
1
Received: comnet
Sent: comnet
Received: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 0110100
1101001
abcdef
Sent: abcdef
Received: 01100001 01100010 01100011 01100100 01100101 01100110
LabsAndAssignments/LabNetworks/Assignment1 on 3: cur_sem [x!?] via 2 v3.9.7 took 16s
λ
```

[0:3.1] 1:zsh- 2:zsh 3:python*

19 Oct 2021 - 06:54 PM - dipesh

-Running server1.py and client1.py side by side

```

LabsAndAssignments/LabNetworks/Assignment1 on ▶ curl_sem [x!?] via 🐍 v3.9.7
+ λ python server2.py
Interface: eth1, Port: 3000, Multi-threaded?: False
Connected to ('10.0.0.1', 60306)
Received: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 011001
01001
Sent: commetsii
Received: 01100011 01101111
Sent: co
Received: 01100011 01101111 01101101 01101110
Sent: comm

LabsAndAssignments/LabNetworks/Assignment1 on ▶ curl_sem [x!?] via 🐍 v3.9.7 took 1m4
6s
λ python client2.py
Interface: eth1, Port: 3000
('10.0.0.1', 3000)
01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001
001
Received: commetsii
01100011 01101111
Sent: 01100011 01101111
Received: co
01100011 01101111 01101101 01101110
Sent: 01100011 01101111 01101101 01101110
Received: comm

```

[0:1.1] 1:python* 2:zsh- 19 Oct 2021 - 06:41 PM - dipesh

- Running server2.py and client2.py

Output (Multi Threaded)

```

LabsAndAssignments/LabNetworks/Assignment1 on ▶ curl_sem [x!?] via 🐍 v3.9.7 took 5s
+ λ python server1.py --multi-threaded
Interface: lo, Port: 3000, Multi-threaded?: True
Connected to ('127.0.0.1', 42526)
Connected to ('127.0.0.1', 42536)
Received: commetsii
Sent: 01100011 01101111 01101101 01100101 01110100 01110011 01101001 01101001
Received: commetsii
Sent: 01100011 01101111 01101101 01100101 01110100 01110011 01101001 01101001
Received: abcd
Sent: 01100001 01100010 01100011 01100100 01100101 01100110
Received: ab
Sent: 01100001 01100010

LabsAndAssignments/LabNetworks/Assignment1 on ▶ curl_sem [x!?] via 🐍 v3.9.7 took 47s
λ python client1.py
Interface: lo, Port: 3000
('127.0.0.1', 3000)
commetsii
Sent: commetsii
Received: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001
101001
abcd
Sent: abcd
Received: 01100001 01100010 01100011 01100100 01100101 01100110
ab
Sent: ab
Received: 01100001 01100010

```

[0:1.2] 1:python* 2:zsh- 3:zsh- 19 Oct 2021 - 07:04 PM - dipesh

- Multiple client1.py connecting to server1.py

```

LabsAndAssignments/LabNetworks/Assignment1 on ✪ cur_sem [x!?] via 🐍 v3.9.7 took 2m1s
* ^ python server2.py --multi-threaded
Interface: eth1, Port: 3000, Multi-threaded?: True
Connected to ('10.0.0.1', 60382)
Connected to ('10.0.0.1', 60384)
Received: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001
Sent: comnetsii
Received: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001
Sent: comnetsii
Received: 01100011 01101111 01101101 01101110 01100101
Sent: come
Received: 01100011 01101111 01101101
Sent: com

LabsAndAssignments/LabNetworks/Assignment1 on ✪ cur_sem [x!?] via 🐍 v3.9.7 took 1m49s
* ^ python client2.py
Interface: eth1, Port: 3000
('10.0.0.1', 3000)
01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001
Sent: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001
01
Received: comnetsii
01100011 01101111 01101101 01101110 01100101
Sent: 01100011 01101111 01101101 01101110 01100101
Received: come

LabsAndAssignments/LabNetworks/Assignment1 on ✪ cur_sem [x!?] via 🐍 v3.9.7 took 1m49s
* ^ python client2.py
Interface: eth1, Port: 3000
('10.0.0.1', 3000)
01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001
Sent: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001
001
Received: comnetsii
01100011 01101111 01101101
Sent: 01100011 01101111 01101101
Received: com

```

[0:1.2] 1:python* 2:zsh- 3:zsh 19 Oct 2021 - 07:06 PM - dipesh

- Multiple client2.py connecting to server2.py

Question 3.a

-> What happens if the client aborts (e.g., when the input is CTRL/D)?

Ans:

When we press Ctrl+D, the client will throw EOFError and exit. The server will see that the client has exited (because the connection will be cut off between them) and hence it is exiting as well .

```
v3.9.7 took 20s
λ python server1.py
Interface: lo, Port: 3000, Multi-threaded?: False
Connected to ('127.0.0.1', 42558)
Received: commetsii
Sent: 01100011 01101111 01101101 01101110 01100101 01110100 01110011
01101001 01101001
LabsAndAssignments/LabNetworks/Assignment1 on ⌘ cur_sem [x!?] via ↵
v3.9.7 took 17s
λ

LabsAndAssignments/LabNetworks/Assignment1 on ⌘ cur_sem [x!?] via ↵
v3.9.7 took 16s
λ python client1.py
Interface: lo, Port: 3000
('127.0.0.1', 3000)
commetsii
Sent: commetsii
Received: 01100011 01101111 01101101 01101110 01100101 01110100 01110011
01101001 01101001
Traceback (most recent call last):
  File "/home/dipesh/Projects/LabsAndAssignments/LabNetworks/Assignment1/client1.py", line 64, in <module>
    main()
  File "/home/dipesh/Projects/LabsAndAssignments/LabNetworks/Assignment1/client1.py", line 50, in main
    msg = input()
EOFError
LabsAndAssignments/LabNetworks/Assignment1 on ⌘ cur_sem [x!?] via ↵
v3.9.7 took 8s
λ
```

- On Ctrl D input

Question 3.b

-> Can you run two clients against the same server? Why or why not?

Hint: Multithreaded socket server in Python

Ans:

This will depend on how we have written the server code. I had written in such a way that we can optionally make the server multi threaded by passing '--multi-threaded' flag while running server1.py or server2.py. If those options are not passed however, it wont have multi-threading.

In the image attached below, we can see that the second instance of client1.py is connecting to the server1.py but the server1.py is not accepting the connection(had it accepted it would've printed it). The same thing would happen for server2.py and client2.py.

If the multi-threaded mode is turned on however, it will handle multiple connections by spawning a thread to handle each connection. It can be seen happening clearly in the Output image for multi-threaded that I have put above in **Output(Multi Threaded)** section above.

```

LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via ℹ v3.9.7 took 9m15s
+ λ python server1.py
Interface: lo, Port: 3000, Multi-threaded?: False
Connected to ('127.0.0.1', 42574)
Received: commetsii
Sent: 01100011 01101111 01101101 01101110 01100101 01110100 01110011 01101001 01101001

LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via ℹ v3.9.7 took 8m58s
λ python client1.py
Interface: lo, Port: 3000
('127.0.0.1', 3000)
commetsii
Sent: commetsii
Received: 01100011 01101111 01101101 01101110 01100101 01110100 01100111 01101001 01101001

LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via ℹ v3.9.7 took 8m53s
λ python client1.py
Interface: lo, Port: 3000
('127.0.0.1', 3000)

LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via ℹ v3.9.7 took 7s
λ

```

- Multiple clients trying to connect to single threaded server1.py

Question 3.c

-> What happens when you try to connect client1 to server2 by passing a wrong network address (e.g., connecting to localhost instead of eth1 IP)?

Ans:

There can be two things that might happen.

Case 1:

If we have server2.py running and we run client1.py , client1.py will try to connect to localhost but localhost wouldnt be running(only eth1 ip would be connectable because server2.py is running). In this case, the result can be seen in the image attached below.

```

LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via 2 v3.9.7 took 8s
λ python server2.py
Interface: eth1, Port: 3000, Multi-threaded?: False

LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via 2 v3.9.7
λ python client1.py
Interface: lo, Port: 3000
Traceback (most recent call last):
  File "/home/dipesh/Projects/LabsAndAssignments/LabNetworks/Assignment1/client1.py", line 65, in <module>
    main()
      File "/home/dipesh/Projects/LabsAndAssignments/LabNetworks/Assignment1/client1.py", line 45, in main
        sock.connect(('0.0.0.0', PORT))
ConnectionRefusedError: [Errno 111] Connection refused
LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via 2 v3.9.7
λ

```

- Case 1

Case 2:

If we pass the eth1 ip to client1.py instead and try to connect to server2.py, the behaviour will be weird because the format in which server2.py expects its message will be different from what its getting. server2.py expects space separated bit strings of length less than or equal to 8 but its getting random ascii strings, so server2.py wont be able to decode those letters and crash as a result.

```

LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via 2 v3.9.7 took 8s
λ python server2.py
Interface: eth1, Port: 3000, Multi-threaded?: False

Connected to ('10.0.0.1', 60526)
Received: commetsii
Traceback (most recent call last):
  File "/home/dipesh/Projects/LabsAndAssignments/LabNetworks/Assignment1/server2.py", line 78, in <module>
    main()
    File "/home/dipesh/Projects/LabsAndAssignments/LabNetworks/Assignment1/server2.py", line 68, in main
      handle_client(conn, addr)
      File "/home/dipesh/Projects/LabsAndAssignments/LabNetworks/Assignment1/server2.py", line 54, in handle_client
        to_be_sent = data_to_send(msg)
        File "/home/dipesh/Projects/LabsAndAssignments/LabNetworks/Assignment1/server2.py", line 43, in data_to_send
          return ''.join(map(lambda x: chr(int(x, 2)), msg.split()))
          File "/home/dipesh/Projects/LabsAndAssignments/LabNetworks/Assignment1/server2.py", line 43, in <lambda>
            return ''.join(map(lambda x: chr(int(x, 2)), msg.split()))
ValueError: invalid literal for int() with base 2: 'commetsii'
LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via 2 v3.9.7 took 5ms
λ
LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via 2 v3.9.7
λ python client1.py --use-ip-addr='10.0.0.1'
Interface: lo, Port: 3000
('10.0.0.1', 3000)
commetsii
Sent: commetsii
Received:

LabsAndAssignments/LabNetworks/Assignment1 on ✘ cur_sem [x!?] via 2 v3.9.7 took 11s
λ

```

- Case 2

Question 1

Code

server1.py

```
import socket
import optparse
import struct
import threading

MAX_QUEUE = 10

'''
This is the part where I am doing
command line argument parsing, so that
the server is more configurable

- The default port is 3000
- The default interface is lo(localhost)
- All these can be overriden through command line args
- To override port number, pass --port <port_no>
- To override interface, pass --interface <interface_name>
'''

parser = optparse.OptionParser()
parser.add_option('-p', '--port', dest='port', default=3000,
                  type="int", help='Port to run the server on')
parser.add_option('-I', '--interface', dest='interface',
                  default='lo', help='Network interface to run the server
on')
parser.add_option('-M', '--multi-threaded', action='store_true',
                  dest='multi_threaded', default=False, help='Tells weather
multi threading should be done to handle multiple requests')

(options, args) = parser.parse_args()
INTERFACE = options.interface
PORT = options.port
MULTI_THREADED = options.multi_threaded
print("Interface: {}, Port: {}, Multi-threaded?: {}".format(INTERFACE,
PORT, MULTI_THREADED))

def pad_to_8chars(c: str) -> str:
```

```
Given a string of length less than
or equal to 8, this will pad it to
8 characters putting 0s in front of it
'''

assert(len(c) <= 8)
return '0'*(8-len(c))+c

def send_int(x: int, conn):
    conn.send(struct.pack("i", socket.htons(x)))

def recv_int(conn) -> int:
    return int.from_bytes(conn.recv(4), byteorder='big')

def data_to_send(msg: str):
    return ''.join((map(lambda x: pad_to_8chars(
        bin(ord(x))[2:])+', ', msg))))


def handle_client(conn, addr):
    print("Connected to ", addr)
    while True:
        data_size = recv_int(conn)
        if(data_size == 0):
            break
        msg = conn.recv(data_size).decode('utf-8')
        print("Received: {}".format(msg))
        to_be_sent = data_to_send(msg)
        send_int(len(to_be_sent), conn)
        conn.send(to_be_sent.encode('utf-8'))
        print("Sent: {}".format(to_be_sent))

def main():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.setsockopt(socket.SOL_SOCKET, 25, INTERFACE.encode())
        sock.bind(('0.0.0.0', PORT))
```

```

        sock.listen(MAX_QUEUE)
        while True:
            if not MULTI_THREADED:
                conn, addr = sock.accept()
                handle_client(conn, addr)
                break
            else:
                conn, addr = sock.accept()
                thrd = threading.Thread(
                    target=handle_client, args=[conn, addr])
                thrd.start()
                continue

main()

```

Question 2

Code

server2.py

```

import socket
import optparse
import struct
import threading

MAX_QUEUE = 10

'''

This is the part where I am doing
command line argument parsing, so that
the server is more configurable

- The default port is 3000
- The default interface is eth1
- All these can be overriden through command line args
- To override port number, pass --port <port_no>

```

```
- To override interface, pass --interface <interface_name>
'''

parser = optparse.OptionParser()
parser.add_option('-p', '--port', dest='port', default=3000,
                  type="int", help='Port to run the server on')
parser.add_option('-I', '--interface', dest='interface',
                  default='eth1', help='Network interface to run the server
on')
parser.add_option('-M', '--multi-threaded', action='store_true',
                  dest='multi_threaded', default=False, help='Tells weather
multi threading should be done to handle multiple requests')
(options, args) = parser.parse_args()
INTERFACE = options.interface
PORT = options.port
MULTI_THREADED = options.multi_threaded
print("Interface: {}, Port: {}, Multi-threaded?: {}".format(INTERFACE,
PORT, MULTI_THREADED))

def send_int(x: int, conn):
    conn.send(struct.pack("i", socket.htons(x)))

def recv_int(conn) -> int:
    return int.from_bytes(conn.recv(4), byteorder='big')

def data_to_send(msg: str):
    return ''.join(map(lambda x: chr(int(x, 2)), msg.split()))

def handle_client(conn, addr):
    print("Connected to ", addr)
    while True:
        data_size = recv_int(conn)
        if(data_size == 0):
            break
        msg = conn.recv(data_size).decode('utf-8')
        print("Received: {}".format(msg))
        to_be_sent = data_to_send(msg)
```

```

        send_int(len(to_be_sent), conn)
        conn.send(to_be_sent.encode('utf-8'))
        print("Sent: {}".format(to_be_sent))

def main():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.setsockopt(socket.SOL_SOCKET, 25, INTERFACE.encode())
        sock.bind(('0.0.0.0', PORT))
        sock.listen(MAX_QUEUE)
        while True:
            if not MULTI_THREADED:
                conn, addr = sock.accept()
                handle_client(conn, addr)
                break
            else:
                conn, addr = sock.accept()
                thrd = threading.Thread(
                    target=handle_client, args=[conn, addr])
                thrd.start()
                continue

main()

```

Question 3

Code

client1.py

```

import socket
import optparse
import struct

'''

This is the part where I am doing

```

```

command line argument parsing, so that
the server is more configurable

- The default port is 3000
- The default interface is lo(localhost)
- All these can be overriden through command line args
- To override port number, pass --port <port_no>
- To override interface, pass --interface <interface_name>
'''

parser = optparse.OptionParser()
parser.add_option('-p', '--port', dest='port', default=3000,
                  type="int", help='Port to run the server on')
parser.add_option('-I', '--interface', dest='interface',
                  default='lo', help='Network interface to run the server
on')
parser.add_option('-a', '--use-ip-addr', dest='ip_addr', default='',
                  help='Connect through IP instead of using the interface')

(options, args) = parser.parse_args()
INTERFACE = options.interface
PORT = options.port
IP_ADDR = options.ip_addr
print("Interface: {}, Port: {}".format(INTERFACE, PORT))

def send_int(x: int, conn):
    conn.send(struct.pack("i", socket.htons(x)))

def recv_int(conn) -> int:
    return int.from_bytes(conn.recv(4), byteorder='big')

def main():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        if(IP_ADDR == ''):
            sock.setsockopt(socket.SOL_SOCKET, 25, INTERFACE.encode())
            sock.connect(('0.0.0.0', PORT))
        else:

```

```

        sock.connect((IP_ADDR, PORT))
        print(sock.getpeername())
        while True:
            msg = input()
            if(msg == ""):
                sock.close()
                break
            msg = msg.strip()
            length_as_bytes = struct.pack("i", socket.htonl(len(msg)))
            sock.send(length_as_bytes)
            sock.send(msg.encode('utf-8'))
            print("Sent: ", msg)
            msg_sz = recv_int(sock)
            recv_msg = sock.recv(msg_sz).decode('utf-8')
            print("Received: ", recv_msg)

main()

```

client2.py

```

import socket
import optparse
import struct

'''

This is the part where I am doing
command line argument parsing, so that
the server is more configurable

- The default port is 3000
- The default interface is eth1
- All these can be overriden through command line args
- To override port number, pass --port <port_no>
- To override interface, pass --interface <interface_name>
'''

```

```
parser = optparse.OptionParser()
parser.add_option('-p', '--port', dest='port', default=3000,
                  type="int", help='Port to run the server on')
parser.add_option('-I', '--interface', dest='interface',
                  default='eth1', help='Network interface to run the server
on')
parser.add_option('-a', '--use-ip-addr', dest='ip_addr', default='',
                  help='Connect through IP instead of using the interface')

(options, args) = parser.parse_args()
INTERFACE = options.interface
PORT = options.port
IP_ADDR = options.ip_addr
print("Interface: {}, Port: {}".format(INTERFACE, PORT))

def send_int(x: int, conn):
    conn.send(struct.pack("i", socket.htons(x)))

def recv_int(conn) -> int:
    return int.from_bytes(conn.recv(4), byteorder='big')

def main():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        if(IP_ADDR == ''):
            sock.setsockopt(socket.SOL_SOCKET, 25, INTERFACE.encode())
            sock.connect(('0.0.0.0', PORT))
        else:
            sock.connect((IP_ADDR, PORT))
    print(sock.getpeername())
    while True:
        msg = input()
        if(msg == ""):
            sock.close()
            break
        msg = msg.strip()
        length_as_bytes = struct.pack("i", socket.htonl(len(msg)))
        sock.send(length_as_bytes)
```

```
    sock.send(msg.encode('utf-8'))
    print("Sent: ", msg)
    msg_sz = recv_int(sock)
    recvd_msg = sock.recv(msg_sz).decode('utf-8')
    print("Received: ", recvd_msg)

main()
```