COMPUTER NETWORKS - 1

CS3530

Assignment - 2

Group members:

CS18BTECH11004 - B CHIRANJEEVI SAI GANESH
CS18BTECH11013 - G VISHAL SIVA KUMAR
CS18BTECH11024 - K VAMSHI KRISHNA REDDY
CS18BTECH11035 - P SAI VARSHITTHA
CS18BTECH11036 - P CHAITANYA JANAKIE
CS18BTECH11045 - T KRISHNA PRASHANTH

Q1)

```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 1$ python3 server.py

Enter name of Server:Server
Connection Established. Connected From: 127.0.0.1
Connection etablished with : Client
Client : hello
Client : This server receives messages from client and sends the same back to it
Client : oh nice
Client : nice to meet you
Client : byee
Client : close
```

```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 1$ python3 client.py  localhost
Connected with Server
Client : hello
Server : hello
Client : This server receives messages from client and sends the same back to it
Server : This server receives messages from client and sends the same back to it
Client : oh nice
Server : oh nice
Client : nice to meet you
Server : nice to meet you
Client : byee
Server : byee
Client : close
```

Here, the hostname of the server which is “localhost” is given as the command line argument for the client. The getaddrinfo() call takes input the hostname of the server, port number and returns corresponding sockaddr structure for that socket. The return value format depends on the address family.

TCP Dump: Here client software sends a DNS query and receives a response.

```
krishna-vm1@krishna-vm1:~$ sudo tcpdump -i ens8
[sudo] password for krishna-vm1:
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens8, link-type EN10MB (Ethernet), capture size 262144 bytes
11:57:30.360480 IP krishna2.example.com.59791 > krishna1.example.com.domain: 41969+ [1au] AAAA? krishna1.exa
11:57:30.360782 IP krishna1.example.com.domain > krishna2.example.com.59791: 41969* 0/1/1 (90)
11:57:30.362168 IP krishna2.example.com.38242 > krishna1.example.com.9999: Flags [S], seq 3538158599, win 64
11:57:30.362189 IP krishna1.example.com.9999 > krishna2.example.com.38242: Flags [S.], seq 899955611, ack 35
gth 0
11:57:30.362882 IP krishna2.example.com.38242 > krishna1.example.com.9999: Flags [.], ack 1, win 502, option
11:57:30.363038 IP krishna2.example.com.38242 > krishna1.example.com.9999: Flags [P.], seq 1:37, ack 1, win
11:57:30.363050 IP krishna1.example.com.9999 > krishna2.example.com.38242: Flags [.], ack 37, win 509, optio
11:57:30.363518 IP krishna1.example.com.9999 > krishna2.example.com.38242: Flags [P.], seq 1:37, ack 37, win
11:57:30.363646 IP krishna1.example.com.9999 > krishna2.example.com.38242: Flags [F.], seq 37, ack 37, win 5
11:57:30.363948 IP krishna2.example.com.38242 > krishna1.example.com.9999: Flags [.], ack 37, win 502, optio
11:57:30.364715 IP krishna2.example.com.38242 > krishna1.example.com.9999: Flags [F.], seq 37, ack 38, win 5
11:57:30.364725 IP krishna1.example.com.9999 > krishna2.example.com.38242: Flags [.], ack 38, win 509, optio
11:57:35.423856 ARP, Request who-has krishna1.example.com tell krishna2.example.com, length 46
11:57:35.423935 ARP, Reply krishna1.example.com is-at 6e:1b:b0:dc:b5:8d (oui Unknown), length 28
11:57:35.602395 ARP, Request who-has krishna2.example.com tell krishna1.example.com, length 28
11:57:35.604471 ARP, Reply krishna2.example.com is-at 52:54:00:b4:0a:c4 (oui Unknown), length 46
^C
16 packets captured
16 packets received by filter
0 packets dropped by kernel
krishna-vm1@krishna-vm1:~$
```

References:

1) https://beej.us/guide/bgnet/html/
2) https://docs.python.org/3/howto/sockets.html
3) https://www.geeksforgeeks.org/socket-programming-python/

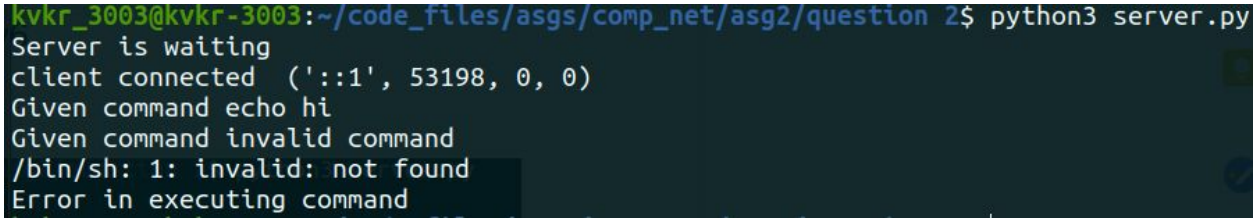
Q2)

<u>Part - 1: Remote Execution of Commands supporting IPv6 and IPv4.</u>

In this feature, When the client sends a command to the server, the server executes it and sends the results back to the client. This feature only supports non interactive commands like “ls”, “echo”, “whoami” etc.

- As of now, we limited the size of output messages sent to client by **2048** chars. So the commands which have large output may not get the desired output.
- This is similar to network protocol like “ssh” where one login to a computer/server from a computer and execute the commands through the shell.
- These programs support both IPv4 and IPv6.

Ipv6 case:



Ipv4 case:

```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 2$ python3 server.py
Server is waiting
client connected  ('::ffff:127.0.0.1', 52176, 0, 0)
Given command ls
Given command echo hi
```

```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 2$ python3 client.py
localhost
want to run a command :
type y for yes
type n for no

enter option(y/n) : y
Enter the cmd : ls
sent  ls
------------------------
output of cmd:
client.py
part 2
server.py
------------------------
enter option(y/n) : y
Enter the cmd : echo hi
sent  echo hi
------------------------
output of cmd:
hi
------------------------
enter option(y/n) : n
```

Here the client sent the commands “ls” and “echo hi” and received the corresponding output.

References:

1) https://docs.python.org/3/library/subprocess.html#subprocess.getstatusoutput
2) https://www.tutorialspoint.com/python/python_networking.htm
3) https://realpython.com/python-sockets/

Part - 2: Supporting IPv6 and IPv4 and Multicast.

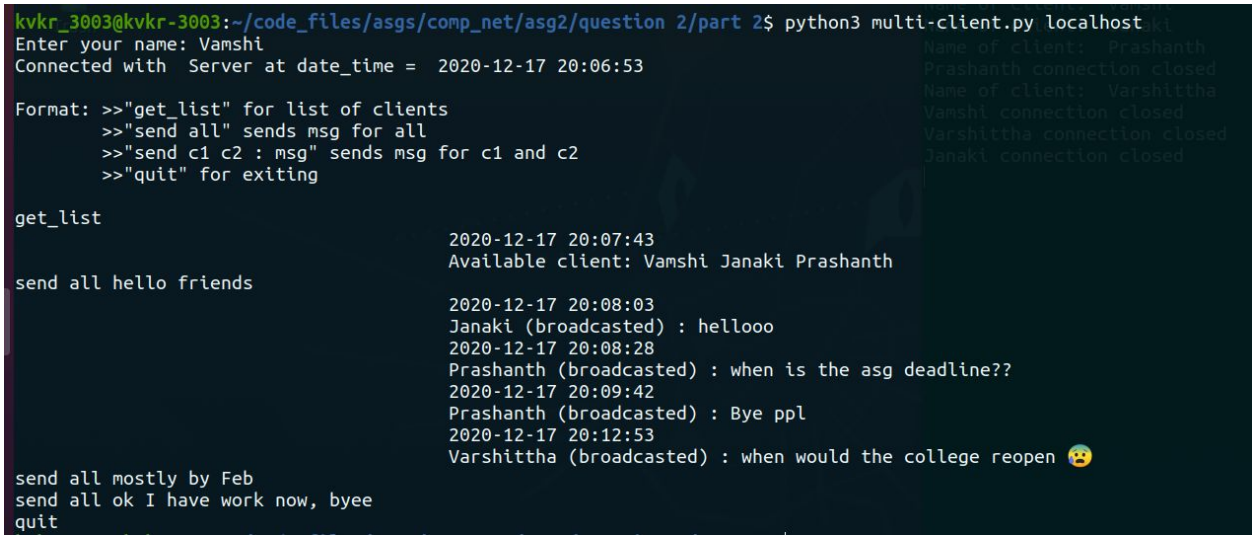
- This feature supports many clients and any client can send messages to any other client. The server manages all the message passing among the clients.

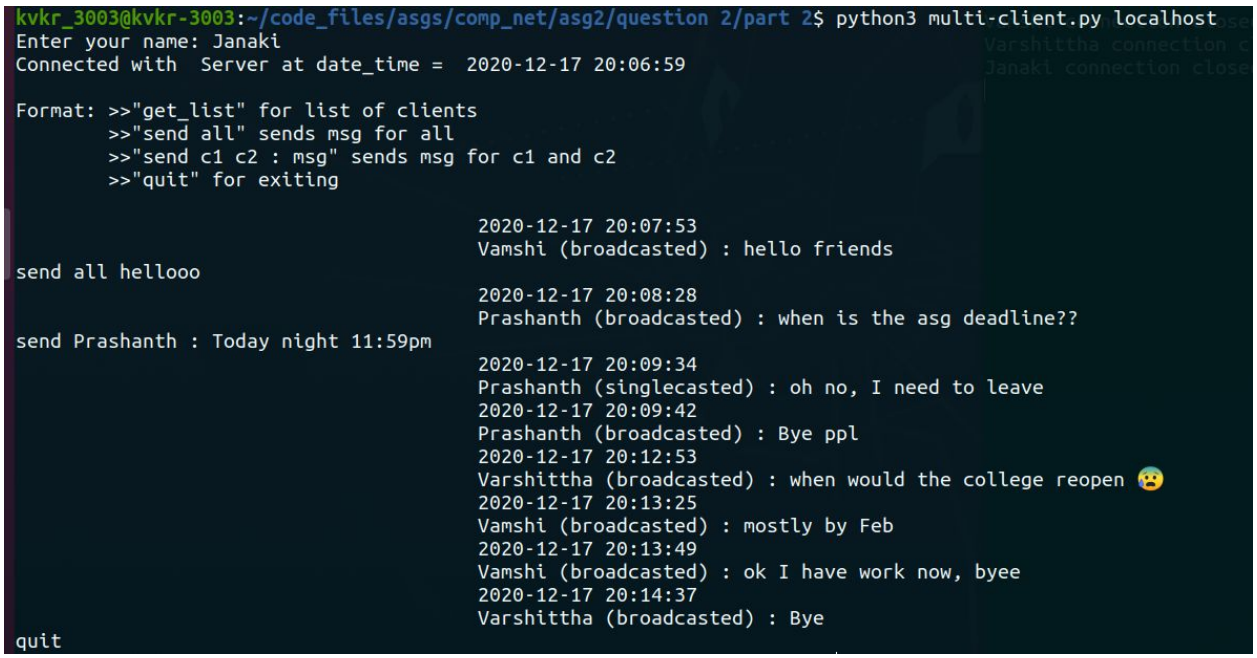
- First the client connects to the server and then the server asks its name. These names are used by the clients to message each other.
- The clients can connect to the server at any time and can also leave at any time they want by sending "**quit**" to the server.
- When a client wants to know the list of users connected to the server, the client can send "**get_list**" to the server and the server sends back the list of users.
- When a client (c1) wants to send a message to another client (c2), c1 should send a msg to the server with a format "**send <c2> : <msg>**". Then the server will send the msg to the client (c2).
- When a client (c1) wants to send a message to all other clients, c1 should send a msg to the server with a format "**send all <msg>**". Then the server will send the msg to all clients except c1.
- When a client (c1) wants to send a message to some other clients (c2, c3, c4), c1 should send a msg to the server with a format "**send c2 c3 c4 : <msg>**". Then the server will send the msg to clients c2, c3, c4.
- To achieve this, we used multithreading so that the server can do message passing between the clients smoothly.
- This is somewhat similar to chatting applications like *messenger*, *whatsapp*.

  Because we can receive all messages on one side and we can run our commands or send messages on the left side
    - Messages sent by single-caste are equivalent to sending private messages to a single person
    - Messages sent by multi-caste are equivalent to sending messages privately to multiple recipients at a time(equivalent to whatsapp forward)
    - Messages sent by broadcast are equivalent to sending group messages
    - Just that we have only 1 terminal for getting input and sending input, we tried our best to give a good interface by classifying the type of messages that each client received
- At the end, if you want to stop the server then you have to do "ctrl+c", otherwise it keeps on waiting for the clients.

Note : Here we also added a **timestamp** for a @message exchange

The following is just a sample to show working of this feature. Where initially three clients connect to the server and send messages to one another. After one client quits, and some other client connects and chat again. At last all quits.





```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 2/part 2$ python3 multi-client.py localhost
Enter your name: Prashanth
Connected with  Server at date_time =  2020-12-17 20:07:06

Format: >>"get_list" for list of clients
        >>"send all" sends msg for all
        >>"send c1 c2 : msg" sends msg for c1 and c2
        >>"quit" for exiting

                                      2020-12-17 20:07:53
                                      Vamshi (broadcasted) : hello friends
                                      2020-12-17 20:08:03
                                      Janaki (broadcasted) : hellooo
send all when is the asg deadline??
                                      2020-12-17 20:09:11
                                      Janaki (singlecasted) : Today night 11:59pm
send Janaki : oh no, I need to leave
send all Bye ppl
quit
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 2/part 2$ python3 multi-client.py localhost
Enter your name: Varshittha
Connected with  Server at date_time =  2020-12-17 20:10:36

Format: >>"get_list" for list of clients
        >>"send all" sends msg for all
        >>"send c1 c2 : msg" sends msg for c1 and c2
        >>"quit" for exiting

send all when would the college reopen 😥
                                      2020-12-17 20:13:25
                                      Vamshi (broadcasted) : mostly by Feb
                                      2020-12-17 20:13:49
                                      Vamshi (broadcasted) : ok I have work now, byee
get_list
                                      2020-12-17 20:14:22
                                      Available client: Janaki Varshittha
send all Bye
quit
```

```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 2/part 2$ python3 mul
ti-server.py
Enter name of Server:Server
server starting at date_time =  2020-12-17 20:06:39
Name of client:  Vamshi
Name of client:  Janaki
Name of client:  Prashanth
Prashanth connection closed
Name of client:  Varshittha
Vamshi connection closed
Varshittha connection closed
Janaki connection closed
^CTraceback (most recent call last):
  File "multi-server.py", line 65, in <module>
    conn, add = s_soc.accept() # accepting connection
  File "/usr/lib/python3.8/socket.py", line 292, in accept
    fd, addr = self._accept()
KeyboardInterrupt
```

References:

1) https://docs.python.org/3/library/socket.html
2) https://medium.com/python-in-plain-english/build-a-chatroom-app-with-python-458fc435025a

3) https://www.geeksforgeeks.org/socket-programming-multi-threading-python/

Q3)

Here we extended the echo server and client by making it protocol independent i.e., it supports both ipv4 and ipv6.

Handling ipv4 address

```
ubuntu@ubuntu ~/D/g/c/question 3 (master)> python3 server.py
Enter name of Server:server
Received connection from  ::ffff:127.0.0.1
Connection Established. Connected From:  ::ffff:127.0.0.1
Connection etablished with : Client
Client : hello
Client : The client received ipv4 address of server
Client : fine..
Client : ok..byee
Client : close
ubuntu@ubuntu ~/D/g/c/question 3 (master)>
```

```
ubuntu@ubuntu ~/D/g/c/question 1 (master)> python3 client.py 127.0.0.1
Connected with server
Client: Hi!!
server : Hi!!
Client: The client received ipv4 address of server
server : The client received ipv4 address of server
Client: fine..
server : fine..
Client: ok..byee
server : ok..byee
Client: close
```

Handling ipv6 address

```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 3$ python3 server.py
Enter name of Server:Server

Received connection from  ::1
Connection Established. Connected From:  ::1
Connection etablished with : Client
Client : Hello
Client : The client received ipv6 address of server
Client : oh great, its handling both ipv4/ipv6
Client : yeah
Client : Ok byee
Client : close
```

```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 3$ python3 client.py ::1
Connected with Server
Client : Hello
Server : Hello
Client : The client received ipv6 address of server
Server : The client received ipv6 address of server
Client : oh great, its handling both ipv4/ipv6
Server : oh great, its handling both ipv4/ipv6
Client : yeah
Server : yeah
Client : Ok byee
Server : Ok byee
Client : close
```

Handling ipv6/ipv4 by giving hostnames instead of addresses

```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 3$ python3 server.py
Enter name of Server:Server
Received connection from  ::ffff:127.0.0.1
Connection Established. Connected From:  ::ffff:127.0.0.1
Connection etablished with : Client
Client : hi
Client : bye
Client : close
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 3$ python3 server.py
Enter name of Server:Server
Received connection from  ::1
Connection Established. Connected From:  ::1
Connection etablished with : Client
Client : Hi
Client : Bye
Client : close
```

```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 3$ python3 client.py localhost
Connected with Server
Client : hi
Server : hi
Client : bye
Server : bye
Client : close
```

```
kvkr_3003@kvkr-3003:~/code_files/asgs/comp_net/asg2/question 3$ python3 client.py ip6-localhost
Connected with Server
Client : Hi
Server : Hi
Client : Bye
Server : Bye
Client : close
```

References:

1) https://medium.com/from-the-scratch/http-server-what-do-you-need-to-know-to-build-a-simple-http-server-from-scratch-d1ef8945e4fa
2) https://www.tutorialspoint.com/python/python_networking.htm