Assignment 4

Siram Nikhil

13th October 2020

1 Question 1

To the traditional Echo server/client model,I added two features.Those are:

1. Reverse shell.
2. Send and get files.(file transfer).

With reverse shell feature, we can connect to the computer which is behind NAT or any firewall. (which is difficult through ssh even if we have ip address). After connecting to the client, we will get total control over a terminal and we can execute any command and get the result of it back to server. You can take screenshot, you can encrypt the files of client without his consent, we can help out friend to solve his/her problems by remotely executing commands etc.
With the file transfer feature, we can send any file to client and execute it using reverse shell feature. We can even send malware and execute. we can send a python file and which on execution send live webcam stream to us. We can also use it for normal large files transfer too.
So these features can help hacker to get whole control of computer as the client.py can be converted to exe file and put on a disk, If a victim just open the disk on laptop the exe file automatically starts running.
These feature can also be used for good purpose. We can help people setup their computer remotely if they face any troubles. We can use it for file transfer etc.

Design and Working of client.py and server.py:

1. Multiple clients can connect to server at same time and then server can choose one from them and start executing commands on that client.
2. There are two threads on server side. One is used for accepting connections and second is used to communicate with clients.
3. there is 'list' command, which will display the clients which are currently online and connected with server.

4. there is 'select' command, which is used to select a particular client.

5. In 'select mode', you can execute any unix/linux commands other than interactive commands like vim, vi etc.
   You can use 'getfile' command to download files from client to server.
   You can use 'sendfile' command to upload a file from server to client.

6. The server is implemented using OOPS concept. It has variables like self.connections and self.addresses which store the connection descriptors and address info of all connected clients and it is accessible to all threads.

```
(base) [siram@siram-Inspiron-5570 q1]$python server.py
ENTER 'help' to get list of commands!

COMM> help
list: prints all online connections
select client_id: selects a client with client_id and allows to send commands
quit: quits current connection with the client you are talking to(to be used when in select mode)
shutdown: shutsdown the server
COMM> list
-----------CLIENTS------------
CLIENT_ID       ADDR

COMM>
Connection has been established with: ('127.0.0.1', 47838)

Connection has been established with: ('127.0.0.1', 47856)
list
-----------CLIENTS------------
CLIENT_ID       ADDR

0: ('127.0.0.1', 47838)

1: ('127.0.0.1', 47856)

COMM> select 1
Now you can send commands to ('127.0.0.1', 47856)!

/home/siram/Downloads/5th_sem/computer_networks/ass4/q1>  cd ..
/home/siram/Downloads/5th_sem/computer_networks/ass4> ls
q1
q2
q3
/home/siram/Downloads/5th_sem/computer_networks/ass4> sendfile README.md
/home/siram/Downloads/5th_sem/computer_networks/ass4> ls
q1
q2
q3
README.md
```

Figure 1: Server talking with a client

Figure 1 and Figure 2 displays demonstration of application and features.

```
/home/siram/Downloads/5th_sem/computer_networks/ass4> gnome-screenshot -f "screen shot"
/home/siram/Downloads/5th_sem/computer_networks/ass4> ls
q1
q2
q3
README.md
screen shot
/home/siram/Downloads/5th_sem/computer_networks/ass4> getfile screen shot
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
DOWNLOADING.........
File downloaded sucessful with name:screen shot

/home/siram/Downloads/5th_sem/computer_networks/ass4> quit
COMM>
Connection has been established with: ('127.0.0.1', 47964)

Please enter valid command!
COMM> shutdown

Quitting gracefully
(base) [siram@siram-Inspiron-5570 q1]$
```

Figure 2: different executions of commands

2 Question 2

Design and Working of Server.py and client.py to construct a multi client communication with single server as middle point of contact:

1. The server and client are implemented using OOPS concept.
2. The multi client facility is achieved by using multi-threading
3. The server object has variables called self.connections and self.addresses which will store connection file descriptors and address info of active clients. these variable are accessible by all threads.
4. main program will create a thread to accept new connections. This thread after accepting new connections, add their info to self variables and also creates new thread for each new connection. The work of these client specific threads is to recv commands,perform required action related to

command received from clients and also send the active connections list to the clients on request.

5. If a client requests server to send a message to other client, then the server will use the target's connection descriptor form self.connections and sends the message as it came from the source client.

6. The client side also has two threads working. One is used to send commands to server, recv the command's output from server and another thread is to listen messages that come from other clients.

7. I have provided "help", "list", "send" commands on the client side.

Figure 3,4,5 displays demonstration of application and features.

```
(base) [siram@siram-Inspiron-5570 q2]$python server.py

Connection has been established with: ('127.0.0.1', 60472)

Connection has been established with: ('127.0.0.1', 60474)
```

Figure 3: server execution

```
siram@siram-Inspiron-5570: ~...   siram@siram-Inspiron-5570: ~...   siram@siram-Inspiron-5570: ~...
(base) [siram@siram-Inspiron-5570 q2]$
(base) [siram@siram-Inspiron-5570 q2]$python client.py
ENTER 'help' TO SEE AVAILABLE COMMANDS!
COMM>help
list: gives list of connections online on server
send client_id msg: will ask server to send 'msg' to client with 'client_id'
quit: to end the connection with server
COMM>
COMM>
COMM>list
-----------CLIENTS-------------
 CLIENT_ID         ADDR
0: ('127.0.0.1', 60472)
1: ('127.0.0.1', 60474)

COMM>send 0 hello,how are you?
MESSAGE sent succefully to ('127.0.0.1', 60472)
COMM>FROM ('127.0.0.1', 60472): I am fine,thanks!

COMM>
```

Figure 4: client_1 execution

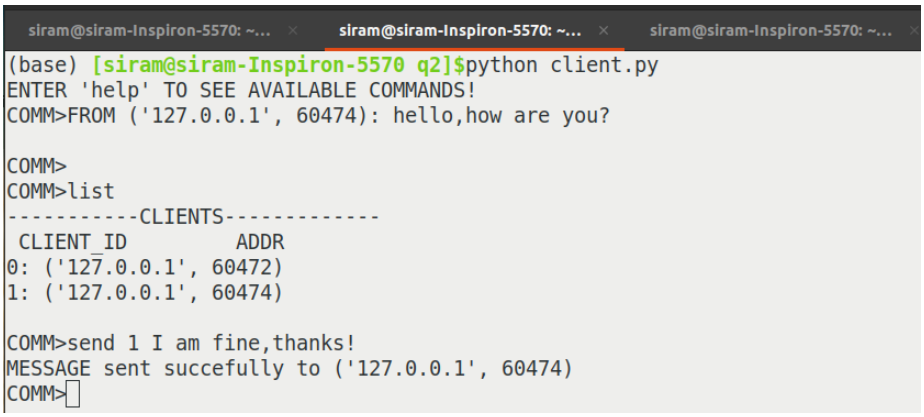

Figure 5: client_2 execution

3 Question 3

To accept both IPV4 and IPV6 connections from clients, we require to open IPV6 socket and set the socket option IPV6_V6ONLY OFF. This allows IPV4 clients to connect to it by IPv4-mapped IPv6 address.
Else we should open two sockets, one for IPV4 and other for IPV6.
I chose option one.
Client gets to choose an optin amongst IPV4 an dIPV6 and give the hostname or ip address.
According to the chosen IP protocol, family will be decide.
If hostname is provided we will be using getaddrinfo() to resolve it to IP address. we will iterate over the results of getaddrinfo to find the correct TRANSPORT PROTOCOL.

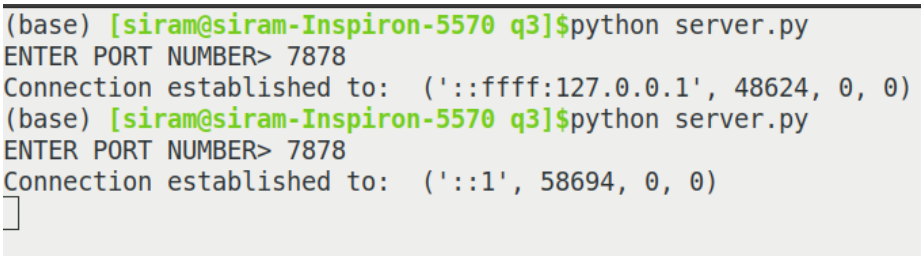

Figure 6: server execution

```
(base) [siram@siram-Inspiron-5570 q3]$python client.py
SELECT IP PROTOCOL TO BE USED:
 1.) IPV4
 2.) IPV6
 ENTER> 1
ENTER HOSTNAME OR IP_ADDR> localhost
ENTER PORT NUMBER> 7878
ENTER 'quit' to exit!
DATA> hello
hello
DATA> hi
hi
DATA> how are u>
how are u>
DATA> quit
```

Figure 7: IPV4 client execution

```
(base) [siram@siram-Inspiron-5570 q3]$python client.py
SELECT IP PROTOCOL TO BE USED:
 1.) IPV4
 2.) IPV6
 ENTER> 2
ENTER HOSTNAME OR IP_ADDR> ip6-localhost
ENTER PORT NUMBER> 7878
ENTER 'quit' to exit!
DATA> hello
hello
DATA> I am fine
I am fine
DATA> quit
```

Figure 8: IPV6 client execution