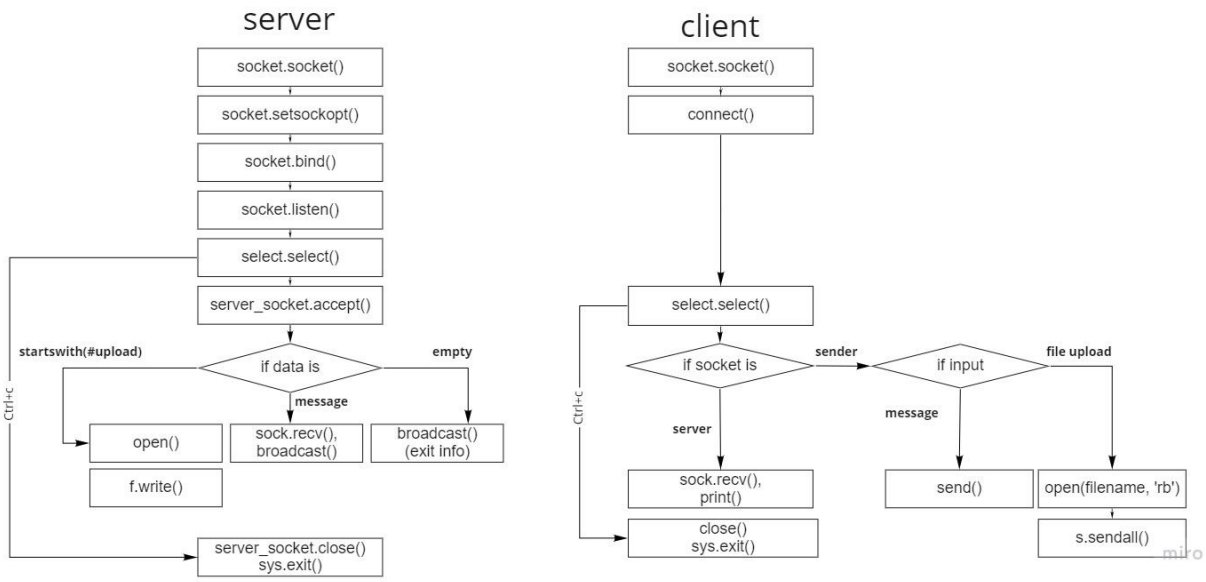
Computer Networks Project 2

2015123080 Jaihong Park

I. Introduction/Reference

- Software environment : ubuntu 20.04, Linux OS
- Programming language : python3
- Reference
  - https://www.bogotobogo.com/python/python_network_programming_tcp_server_client_chat_server_chat_client_select.php
  - https://www.thepythoncode.com/article/send-receive-files-using-sockets-python
  - https://docs.python.org/3/library/socket.html#socket.socket.listen

II. Flow Chart



III. Explanation of 7 functions

1. **socket(socket.AF_INET, socket.SOCK_STREAM)**
   This function creates a new socket using AF_INET as address and the type is SOCK_STREAM. AF_INET is a address family, contains (host, port). Host is a string that has either a hostname or IPv4 address, port is a PORT number which is an integer. SOCK_STREAM is a socket type used for TCP protocol.

2. **setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)**
   This function sets options on sockets. SOL_SOCKET is representing sockets API level, SO_REUSEADDR flag is used for reusing a local socket in TIME_WAIT state. 1 represents the buffer.

3. **bind((HOST, PORT))**
   This function binds the socket to address.

4. **listen(n)**
   This function enables a server to accept connections. n is the backlog which specifies the number of unaccepted connections that the system will allow before refusing new connection.

5. **connect((HOST,PORT))**
   This function connects to a remote socket at given address.

6. **accept()**
   This function is used to accept a connection and returns a pair (conn, address). conn is the new socket object, address is a pair that contains address of the new socket. Before accept(), the socket must be bound to and address, and listening for connections.

7. **close()**
   This function closes the socket and deallocates its resources.

IV. Logical Explanations

I will first explain srv.py and then cli.py.

1. **Create server socket**

```
def chat_server():

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((HOST, PORT))
    server_socket.listen(10)
    client_count = 0
    # add server socket object to the list of readable connections
    SOCKET_LIST.append(server_socket)

    print("Chat server started on port " + str(PORT) +".")
```

Create socket using socket API. Functions are explained in part 3. client_count is used. SOCKET_LIST is the list of readable connections.

```
HOST = sys.argv[1]
PORT = int(sys.argv[2])
SOCKET_LIST = []
RECV_BUFFER = 4096
```

Get hostname and port number by reading command line arguments.

```
if(len(sys.argv) < 3) :
    print('valid format : python3 cli.py hostname port')
    sys.exit()
```

Check if argument is in valid format

```
#make directory
createFolder(os.getcwd()+"/server")
```

Create ‘server’ directory in the current working directory. I used os.getcwd() to get current working directory.

```
def createFolder(directory):
    try:
        if not os.path.exists(directory):
            os.makedirs(directory)
    except OSError:
        print ('Error: Creating directory. ' +  directory)
```

Create folder if it does not exist with os.path.exists() and os.makedirs()

2. **Run the server, monitor all sockets with select()**

```
while 1:
    try:
        # get the list sockets by select()
        ready_to_read,ready_to_write,in_error = select.select(SOCKET_LIST,[],[])
```

```
server_socket.close()
```

Run the server. I used select() to handle multiple clients instead of multi-threading.
Select() function monitors all the client sockets and server socket which are readable. When client sends a message, it is a readable socket. Select() works by blocking. I did not use timeout.
When while() ends, close socket.

3. **New client connection**

```
for sock in ready_to_read:
    # a new client connection
    if sock == server_socket:
        sockfd, addr = server_socket.accept()
        SOCKET_LIST.append(sockfd)
        client_count += 1
```

If server is readable, it means there is a new client connection.

sockfd is the client socket, addr is the tuple that contains address and port number of the client.

Append the socket to the socket list and add 1 to client count.

```
#print, broadcast the connection info
if client_count > 1:
    print("> New user %s:%s entered " % addr + "(%d users online)" % client_count)
    sockfd.send( ("> Connected to the chat server " + "(%d users online)" % client_count).encode())
    broadcast(server_socket, sockfd, "> New user %s:%s entered " % addr + "(%d users online)" % client_count)
else:
    print("> New user %s:%s entered " % addr + "(%d user online)" % client_count)
    sockfd.send( ("> Connected to the chat server " + "(%d user online)" % client_count).encode())
    broadcast(server_socket, sockfd, "> New user %s:%s entered " % addr + "(%d user online)" % client_count)
```

Print the connection info on the server and clients' terminal. It contains three part. Server, Sender, and the rest. Send message to the sender. I defined broadcast() function to send messages to clients except the sender.

If there are more than 2 users, append letter 's' to 'user'.

```
# broadcast chat messages to all connected clients
def broadcast (server_socket, sock, message):
    for socket in SOCKET_LIST:
        # send the message only to peer
        if socket != server_socket and socket != sock :
            socket.send(message.encode())
```

Broadcast function() is used for sending message only to peer. Gets server_socket, sender socket, message as parameters. Send only to peers of the sender. I converted message to byte object with encode().

4. **Get message from a client**

```
# a message from a client
else:
        # receive data from the socket.
        data = sock.recv(RECV_BUFFER)
        # print, broadcast data
        if data:
            print("[%s:%s] " % sock.getpeername() + data.decode() )
            broadcast(server_socket, sock, "[%s:%s] " % sock.getpeername() + data.decode())
```

If client socket is readable, it means that a client sent a message.
Receive data by recv(), RECV_BUFFER is 4096 bytes.
If data is not empty, send message to all the sockets.
Data is a byte-object, so converted data to string by decode().
I used getpeername() to get IP address and Port number.

5. **File uploaded by a client**

```
#file uploading
if data.decode().startswith('#upload'):
    #get filename, filesize
    command, filename, filesize = data.decode().split()
    #make complete filename
    folder_path = os.getcwd() + "/server"
    file_name = filename.replace("'","")
    complete_name = os.path.join(folder_path,file_name)
```

If data starts with '#upload', uploading process begins.
Get filename and filesize with decode().split() function
Make complete filename with 'os' library. Remove single quotes with replace().
I used os.path.join() to make complete filename. (~/server/filename.xx)

```
#read file to the end.
total_bytes_read = 0
file_size = int(filesize)
with open(complete_name, "wb") as f:
    while total_bytes_read < file_size:
        data = sock.recv(4092)
        total_bytes_read += len(data)
        f.write(data)
f.close()
```

Read and write file under the 'server' folder. I used total_bytes_read to end the loop. When total_bytes_read >= file_size, file transfer is completed.
read file with recv() and write it on complete_name with f.write().

```
#send upload info
print("> User %s:%s has uploaded a file" % sock.getpeername())
broadcast(server_socket, sock, "> User %s:%s has uploaded a file" % sock.getpeername())
```

Send message of upload info to peers.

6. **Remove broken socket**

```
# remove broken socket
else:
    if sock in SOCKET_LIST:
        SOCKET_LIST.remove(sock)
    # print, broadcast the exit info
    client_count -= 1
    if client_count > 1:
        print("< The user %s:%s left " % sock.getpeername() + "(%d users online)" % client_count)
        broadcast(server_socket, sock, "< The user %s:%s left " % sock.getpeername() + "(%d users online)" % client_count)
    else:
        print("< The user %s:%s left " % sock.getpeername() + "(%d user online)" % client_count)
        broadcast(server_socket, sock, "< The user %s:%s left " % sock.getpeername() + "(%d user online)" % client_count)
```

I considered the connection is closed when the readable socket data is empty.
Subtract 1 from client_count. Print message on the server and send messages to peers

7. **Terminate the server socket**

```
# terminate when ctrl+c pressed
except KeyboardInterrupt:
    sys.stdout.write("\033[F")
    print("\nexit")
    server_socket.close()
    sys.exit()
```

When ctrl+c is pressed KeyboardInterrupt is signaled
Terminate the server socket with printing ‘exit’.
I used “\033[F” to delete ^C, and print “exit”.
Close server socket and terminate.

Now I will explain the client.py

8. **Create client socket and connect to the server**

```
#check argurments
def chat_client():
    if(len(sys.argv) < 3) :
        print('valid format : python3 cli.py hostname port')
        sys.exit()
```

first check arguments

```
#server IP and Port#
host = sys.argv[1]
port = int(sys.argv[2])
```

Get IP and port# from command line arguments

```
#create socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# connect to remote host
try :
    s.connect((host, port))

except :
    print('Unable to connect')
    sys.exit()
```

Create socket with socket.socket() and connect to the server with connect()

9. Start chatting

```
while 1:

    # Get the list sockets which are readable
    read_sockets, write_sockets, error_sockets = select.select([sys.stdin, s] , [], [])
```

I used select() to monitor server and client sockets. [sys.stdin, s] is the socket list. There are no timeout.

10.Receive message from server

```
for sock in read_sockets:
    #if readable socket is server
    if sock == s:
        # get data from server
        data = sock.recv(4096)
        if data:
            #print data
            print("\r"+data.decode())
            sys.stdout.write('[You] '); sys.stdout.flush()
```

If readable socket is server, get message by recv(), print it with decode().
To print received message before '[You]', I used '\r' in the front.
To print without a new line after '[You]', I used sys.stdout.write()

```
# terminate socket if the server is terminated
else:
    print("\n server closed")
    s.close()
    sys.exit()
```

If data is empty, I considered it as termination of the server.
When server is terminated, socket will be closed.

11.Upload file or send message to the server

```
else:
    #read message
    message = input()
```

Read message from user.

```
if message=="I will upload a file":
    s.send("I will upload a file".encode())
```

If user types “I will upload a file”, uploading process begins.
I sent this message to the server as well.

```
#get filename and compute filesize
command, filename = sys.stdin.readline().split()
filesize = os.path.getsize(filename.replace("'",""))
#send command, filename, filesize to server
s.send((command+" "+filename+" "+str(filesize)).encode())
```

Get filename from sys.stdin.readline().split() (#upload ‘filname.xx’)
Compute filesize with os.path.getsize(), and send these information to the server.
I combined these information into one string with white space between each element, and encode() to convert string into byte object.

```
#read file and send it to server
f = open(filename.replace("'",""),'rb')
l = f.read(4096)
while l:
    s.sendall(l)
    l = f.read(4096)
f.close()
```

Read file with open(). and send it to the server with sendall()
I used sendall() since it sends the entire buffer or throws an exception.
send() can send less bytes than I requested.

```
#print upload info, then go back to chatting
sys.stdout.write("\033[F")
sys.stdout.write("\r> File %s has uploaded!\n" % filename)
sys.stdout.write('[You] '); sys.stdout.flush()
```

Print upload info. I used “\033[F” to clear the previous command line. Then go back to chatting

```
else:
    #send message to the server
    s.send(message.encode())
    sys.stdout.write('[You] '); sys.stdout.flush()
```

If it is just message, send message to the server. I used input().encode() to convert string into byte object.

12. Terminate client socket

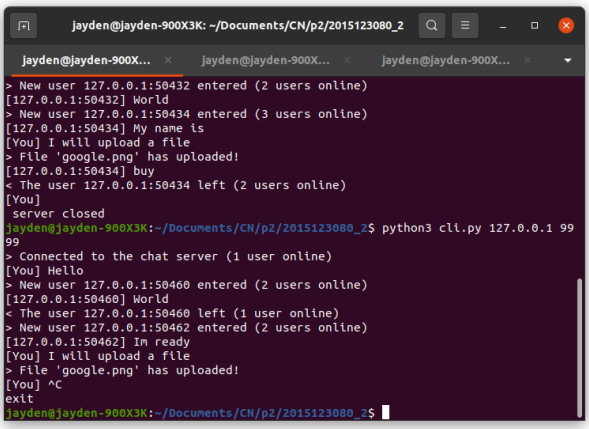
```
#terminate when ctrl+c pressed
except KeyboardInterrupt:
    print("\nexit")
    s.close()
    sys.exit()
```

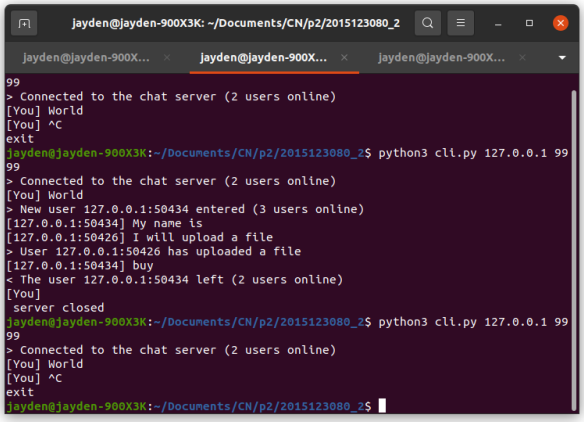
When ctrl+c is pressed, close socket and terminate. I couldn't delete ^C. print "exit"

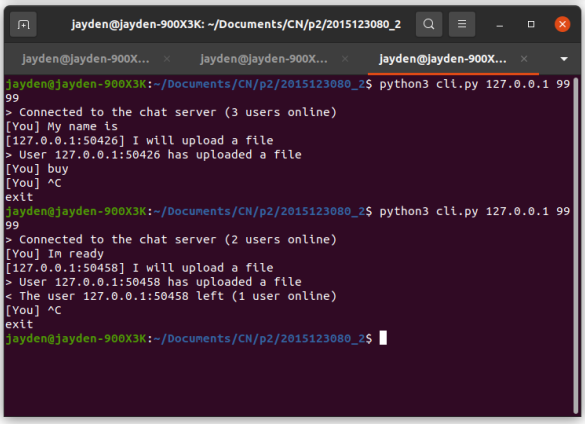
```
import sys, socket, select,os
```

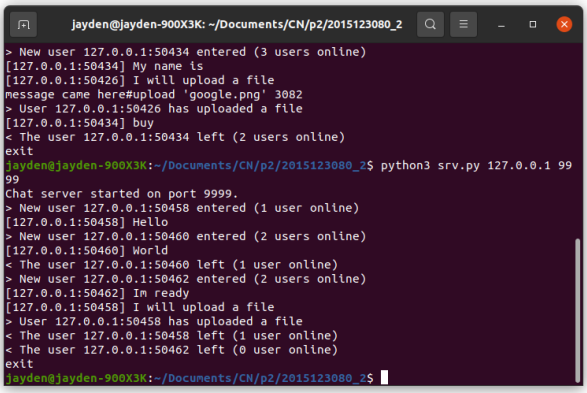
Both srv.py and cli.py import same libraries

V. Snapshots


jayden@jayden-900X3K: ~/Documents/CN/p2/2015123080_2
jayden@jayden-900X...
jayden@jayden-900X...
jayden@jayden-900X...
> New user 127.0.0.1:50432 entered (2 users online)
[127.0.0.1:50432] World
> New user 127.0.0.1:50434 entered (3 users online)
[127.0.0.1:50434] My name is
[You] I will upload a file
> File 'google.png' has uploaded!
[127.0.0.1:50434] buy
< The user 127.0.0.1:50434 left (2 users online)
[You]
server closed
jayden@jayden-900X3K:~/Documents/CN/p2/2015123080_2$ python3 cli.py 127.0.0.1 99
99
> Connected to the chat server (1 user online)
[You] Hello
> New user 127.0.0.1:50460 entered (2 users online)
[127.0.0.1:50460] World
< The user 127.0.0.1:50460 left (1 user online)
> New user 127.0.0.1:50462 entered (2 users online)
[127.0.0.1:50462] Im ready
[You] I will upload a file
> File 'google.png' has uploaded!
[You] ^C
exit
jayden@jayden-900X3K:~/Documents/CN/p2/2015123080_2$


jayden@jayden-900X3K: ~/Documents/CN/p2/2015123080_2
jayden@jayden-900X...
jayden@jayden-900X...
jayden@jayden-900X...
99
> Connected to the chat server (2 users online)
[You] World
[You] ^C
exit
jayden@jayden-900X3K:~/Documents/CN/p2/2015123080_2$ python3 cli.py 127.0.0.1 99
99
> Connected to the chat server (2 users online)
[You] World
> New user 127.0.0.1:50434 entered (3 users online)
[127.0.0.1:50434] My name is
[127.0.0.1:50426] I will upload a file
> User 127.0.0.1:50426 has uploaded a file
[127.0.0.1:50434] buy
< The user 127.0.0.1:50434 left (2 users online)
[You]
server closed
jayden@jayden-900X3K:~/Documents/CN/p2/2015123080_2$ python3 cli.py 127.0.0.1 99
99
> Connected to the chat server (2 users online)
[You] World
[You] ^C
exit
jayden@jayden-900X3K:~/Documents/CN/p2/2015123080_2$


jayden@jayden-900X3K: ~/Documents/CN/p2/2015123080_2
jayden@jayden-900X...
jayden@jayden-900X...
jayden@jayden-900X...
jayden@jayden-900X3K:~/Documents/CN/p2/2015123080_2$ python3 cli.py 127.0.0.1 99
99
> Connected to the chat server (3 users online)
[You] My name is
[127.0.0.1:50426] I will upload a file
> User 127.0.0.1:50426 has uploaded a file
[You] buy
[You] ^C
exit
jayden@jayden-900X3K:~/Documents/CN/p2/2015123080_2$ python3 cli.py 127.0.0.1 99
99
> Connected to the chat server (2 users online)
[You] Im ready
[127.0.0.1:50458] I will upload a file
> User 127.0.0.1:50458 has uploaded a file
< The user 127.0.0.1:50458 left (1 user online)
[You] ^C
exit
jayden@jayden-900X3K:~/Documents/CN/p2/2015123080_2$


jayden@jayden-900X3K: ~/Documents/CN/p2/2015123080_2
> New user 127.0.0.1:50434 entered (3 users online)
[127.0.0.1:50434] My name is
[127.0.0.1:50426] I will upload a file
message came here#upload 'google.png' 3082
> User 127.0.0.1:50426 has uploaded a file
[127.0.0.1:50434] buy
< The user 127.0.0.1:50434 left (2 users online)
exit
jayden@jayden-900X3K:~/Documents/CN/p2/2015123080_2$ python3 srv.py 127.0.0.1 99
99
Chat server started on port 9999.
> New user 127.0.0.1:50458 entered (1 user online)
[127.0.0.1:50458] Hello
> New user 127.0.0.1:50460 entered (2 users online)
[127.0.0.1:50460] World
< The user 127.0.0.1:50460 left (1 user online)
> New user 127.0.0.1:50462 entered (2 users online)
[127.0.0.1:50462] Im ready
[127.0.0.1:50458] I will upload a file
> User 127.0.0.1:50458 has uploaded a file
< The user 127.0.0.1:50458 left (1 user online)
< The user 127.0.0.1:50462 left (0 user online)
exit
jayden@jayden-900X3K:~/Documents/CN/p2/2015123080_2$