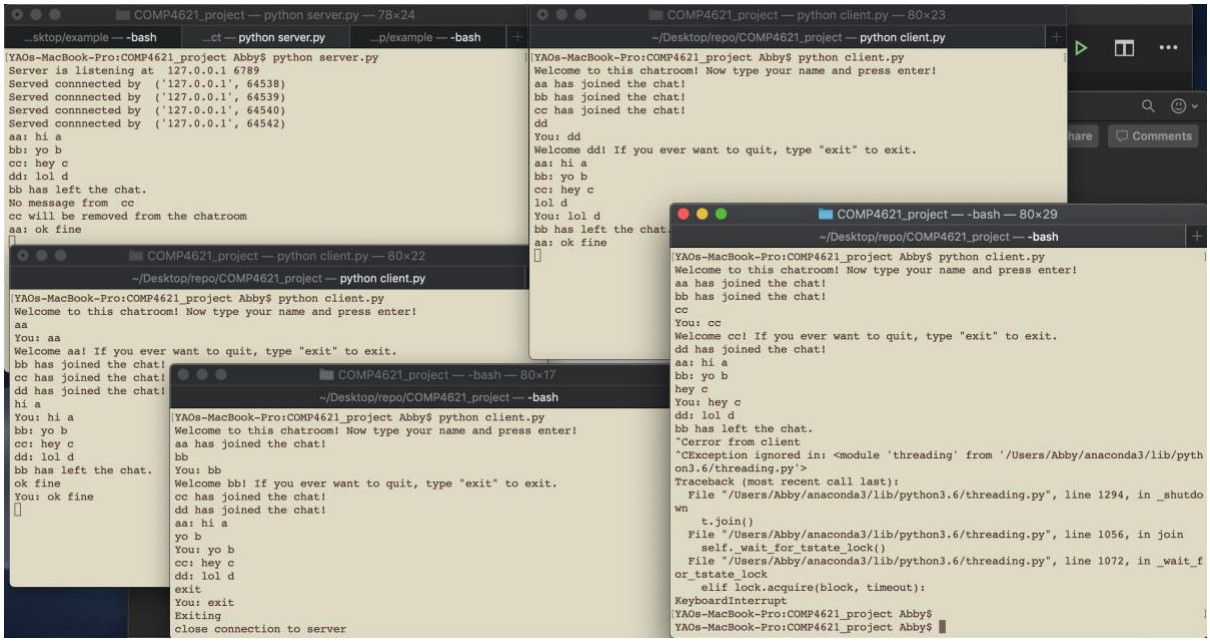
COMP4621 Fall2019 Project Report

YAO Xinjie(20327521) xyaoab@connect.ust.hk

Python 3.6
IDE: Visual Studio Code
Test scenario:

1. one server + four clients (aa,bb,cc,dd)
2. bb enters exit to close connection – broadcast 'bb has left the chat'
3. cc loses connection by keyboard interruption(ctrl-C) – showing 'no message from cc, cc will be removed from the chatroom'



Implementation Details:

1. Multi-clients:
   Listofclients store all connected clients, message will be sent to all other clients' sockets excluding itself

```
#conn is client itself
def broadcast(msg, conn):
    #broadcast the message to other clients except the client who sent it
    # exclude it self
    for socket in listofclients:
        if socket != conn:
            try:
                socket.send(msg)
            except:
                #broken socket ended by ctrl+c
                print('[Error]Broken socket ', socket, ' closing connection!')
                socket.close()
                remove(socket)
```

   Remove clients who close the socket from listfoclients

```
def remove(conn):
    #remove connection if the user wants to exit or the connection is interrupted
    if conn in listofclients:
        listofclients.remove(conn)
```

2. Reliability
   - Use TCP connection indicated by socket_stream

```
# setup server
PORT = 6789
HOST = socket.gethostbyname('localhost')
listofclients=[]
serverAddr = (HOST, PORT)
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
serverSocket.bind(serverAddr)
```

   - Try-except to catch error message whenever receiving or sending blank message (exit without notification)

```
while True:
    # receive message and print it
    try:
        message = connSocket.recv(1024)
        #accidentally lose connection -ctrl-c
        if not message:
            print("No message from ",name)
            print("%s will be removed from the chatroom" %name)
            break
```

3. Exit:
   - Close connection for client and server socket
   - Break the while loop
   - Remove itself from listofclients

```
if message.decode()=='exit':
    #connSocket.send(message)
    msg = name + ' has left the chat.'
    print(msg)
    connSocket.close()
    remove(connSocket)
    broadcast(msg.encode(),connSocket)
    break
```

```
def handle_server_message(connSocket):
    while True:
        try:
            message = connSocket.recv(1024)
            if not message:
                print("close connection to server")
                break
            print(message.decode())
        except:
            print(connSocket, "error received from server")
            break
    connSocket.close()

# start a new thread to receive message
threading.Thread(target = handle_server_message, args = (clientSocket,)).start()

while True:
    # input something and send it to server
    try:
        inputMessage = input()
        clientSocket.send(inputMessage.encode())
        print("You: "+inputMessage)
        # handle exit
        if inputMessage=='exit':
            print('Exiting')
            break
    except:
        print("error from client")
        break
```

4. Exception handling
   - E.g. Ctrl-C client.py –detect blank message and break the while loop

```
while True:
    # receive message and print it
    try:
        message = connSocket.recv(1024)
        #accidentally lose connection -ctrl-c
        if not message:
            print("No message from ",name)
            print("%s will be removed from the chatroom" %name)
            break
        # handle exit
```

   - Whenever break the while loop, close socket connection

```
        except:
            print("no response, closing connection from ", connSocket)
            break
    connSocket.close()
    remove(connSocket)
```

5. Scalability

   Problems:
   - Computationally costly to run a large of threads and establish sockets for each
   - Resource competition leads to deadlock, e.g. simultaneous push message, which may cause message loss
   - Message delay due to concurrency – can't ensure real-time messaging
   - Store a list of clients – memory costly, searching is time-consuming

Solutions:

- Maintaining a list of active clients given not all clients will be online actively
    - Drop connections after a certain period of time
    - Reconnect when clients send message again
- Use mutex to lock the sending and receiving process
    - When a client is sending, others message will be cached to send out once it finishes
    - When a client is receiving, the new coming message will be queued
- Have a maximum buffer size to cache received/sent messages in the server
    - Avoid clients flushing the server too much