COL334 ASSIGNMENT 2 REPORT

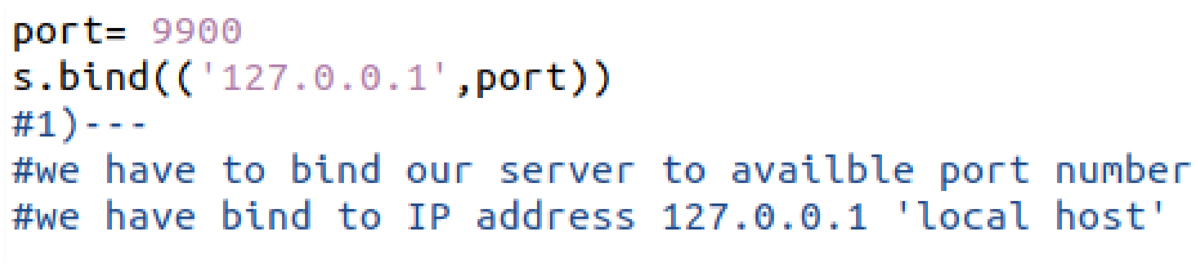
Gaurav Kumar

Entry No: 2019CS50430

1 Design Decisions At Server Side

1.1

We have bind our server to available port number(eg-9990) and at local-host IP i.e '127.0.0.1'

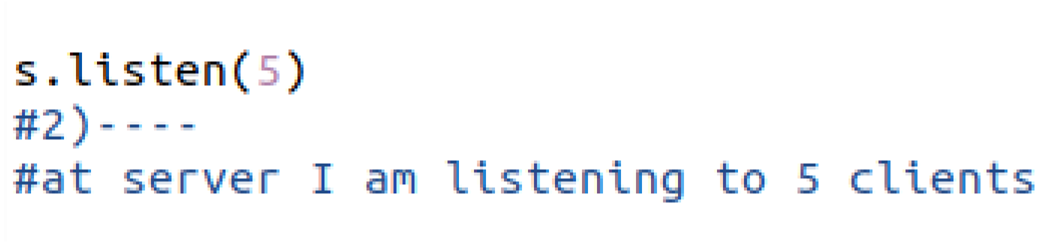


```
port= 9900
s.bind(('127.0.0.1',port))
#1)---
#we have to bind our server to availble port number
#we have bind to IP address 127.0.0.1 'local host'
```

Figure 1:

1.2

Server is listening to 5 clients

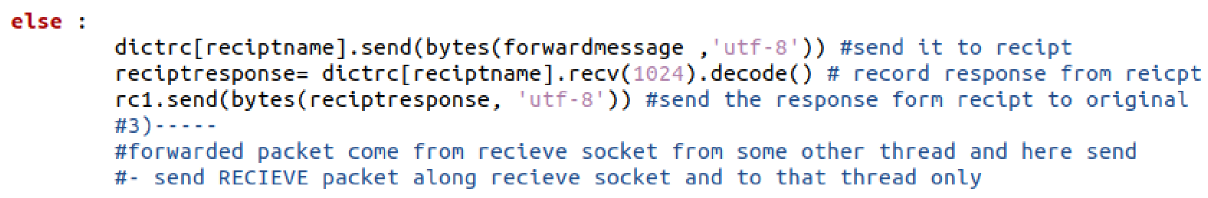


```
s.listen(5)
#2)----
#at server I am listening to 5 clients
```

Figure 2:

1.3

If a client recieve a forwarded packet from its reciving socket he should RECIEVE message from that recieving socket only

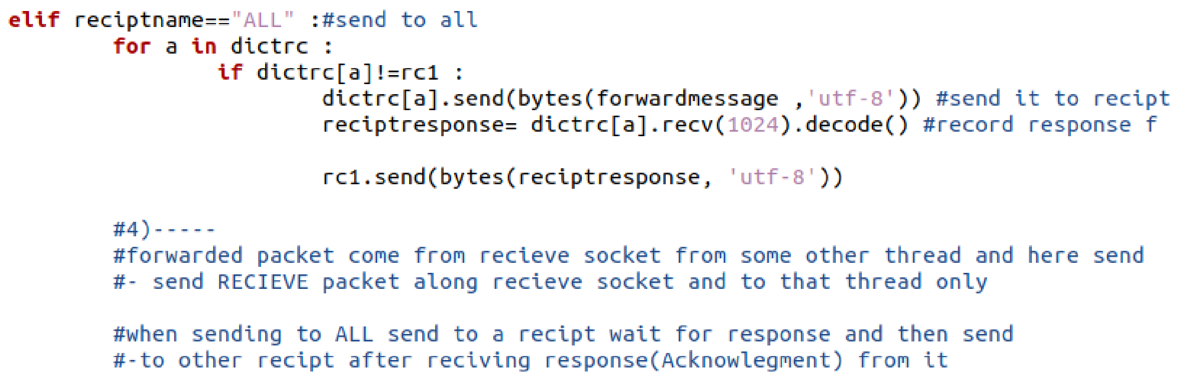


```
 else :
        dictrc[reciptname].send(bytes(forwardmessage ,'utf-8')) #send it to recipt
        reciptresponse= dictrc[reciptname].recv(1024).decode() # record response from reicpt
        rc1.send(bytes(reciptresponse, 'utf-8')) #send the response form recipt to original
        #3)-----
        #forwarded packet come from recieve socket from some other thread and here send
        #- send RECIEVE packet along recieve socket and to that thread only
```

Figure 3:

1.4

When u are sending to all recipt , u send to receipt and wait for his response and after getting its response send to next receipt in the list

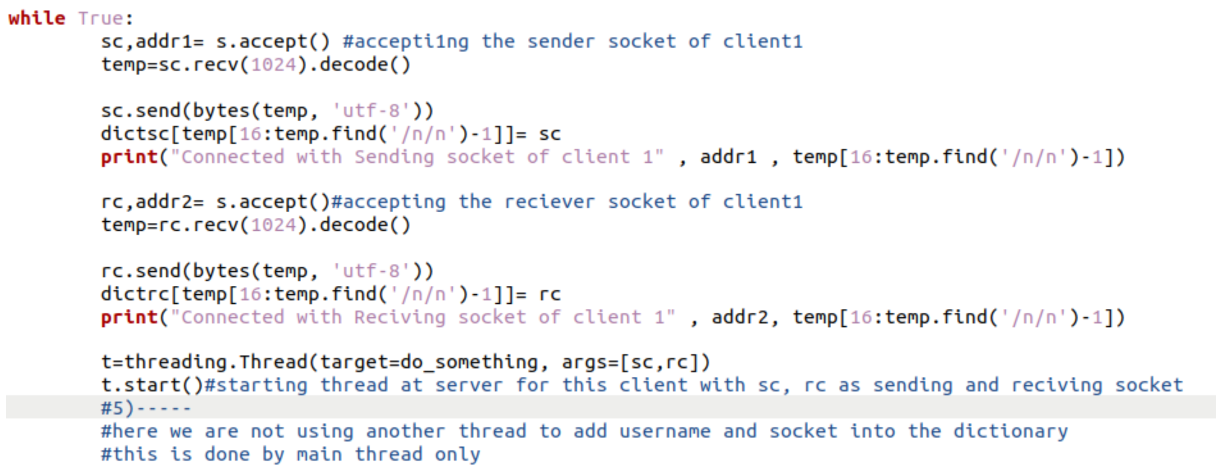


```
elif reciptname=="ALL" :#send to all
        for a in dictrc :
                if dictrc[a]!=rc1 :
                        dictrc[a].send(bytes(forwardmessage ,'utf-8')) #send it to recipt
                        reciptresponse= dictrc[a].recv(1024).decode() #record response f

                        rc1.send(bytes(reciptresponse, 'utf-8'))

        #4)-----
        #forwarded packet come from recieve socket from some other thread and here send
        #- send RECIEVE packet along recieve socket and to that thread only

        #when sending to ALL send to a recipt wait for response and then send
        #-to other recipt after reciving response(Acknowlegment) from it
```

Figure 4:

1.5

Here we are not using some other thread to add username and sockets to our dictionary it is done by our MAIN thread only



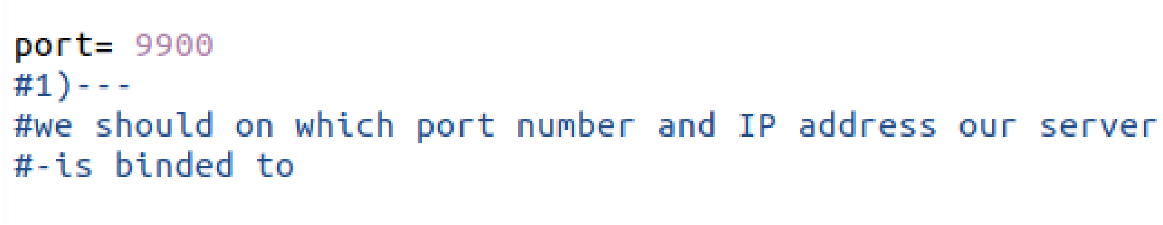
```
while True:
        sc,addr1= s.accept() #accepti1ng the sender socket of client1
        temp=sc.recv(1024).decode()

        sc.send(bytes(temp, 'utf-8'))
        dictsc[temp[16:temp.find('/n/n')-1]]= sc
        print("Connected with Sending socket of client 1" , addr1 , temp[16:temp.find('/n/n')-1])

        rc,addr2= s.accept()#accepting the reciever socket of client1
        temp=rc.recv(1024).decode()

        rc.send(bytes(temp, 'utf-8'))
        dictrc[temp[16:temp.find('/n/n')-1]]= rc
        print("Connected with Reciving socket of client 1" , addr2, temp[16:temp.find('/n/n')-1])

        t=threading.Thread(target=do_something, args=[sc,rc])
        t.start()#starting thread at server for this client with sc, rc as sending and reciving socket
        #5)-----
        #here we are not using another thread to add username and socket into the dictionary
        #this is done by main thread only
```

Figure 5:

2 Design Decisions At Client Side

2.1

We should on which port number and IP address our server is currently binded to

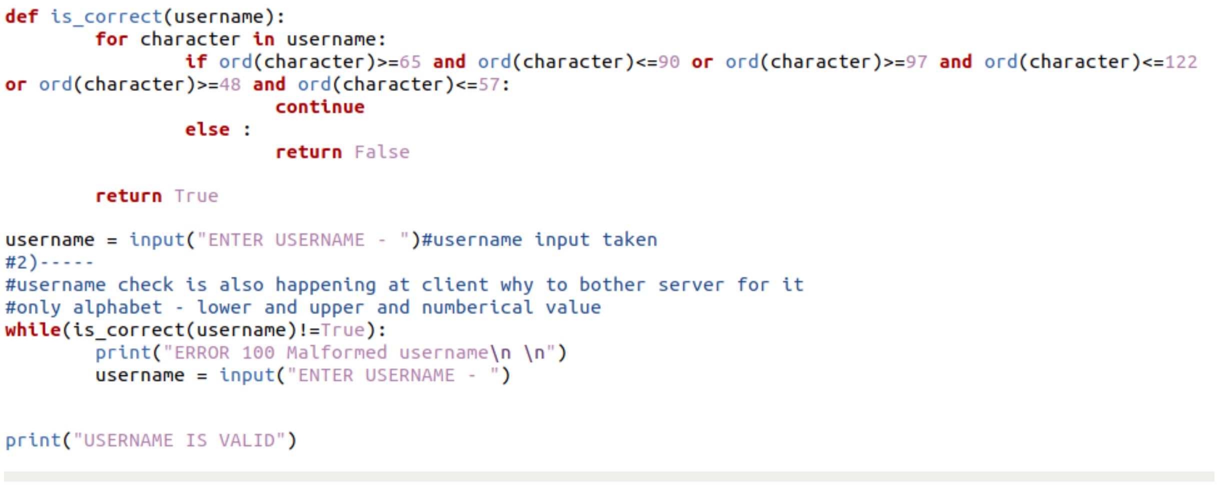


```
port= 9900
#1)---
#we should on which port number and IP address our server
#-is binded to
```

Figure 6:

2.2

Here username check is done at the client side only why to bother server for this

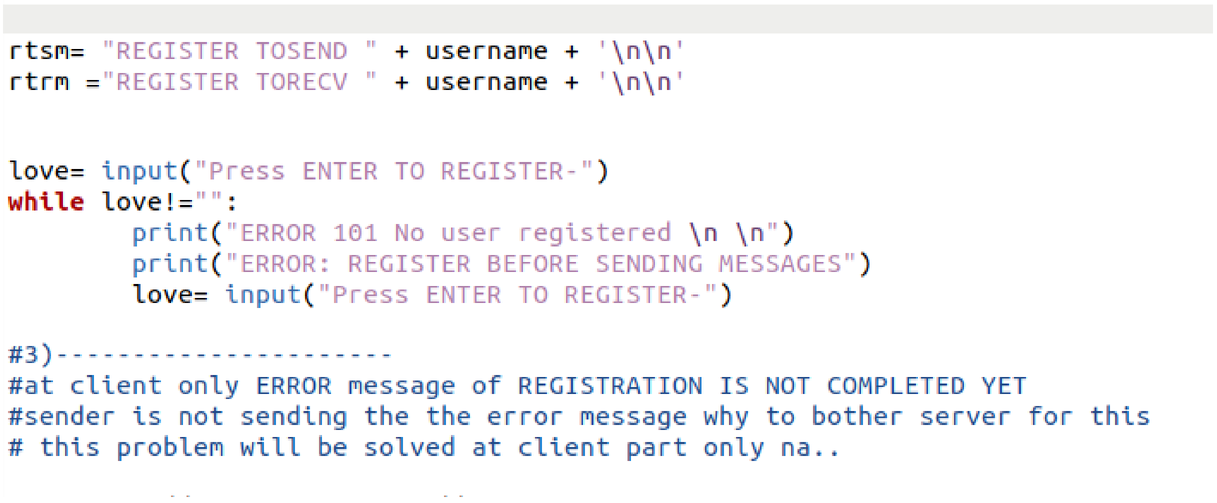


```
def is_correct(username):
        for character in username:
                if ord(character)>=65 and ord(character)<=90 or ord(character)>=97 and ord(character)<=122
or ord(character)>=48 and ord(character)<=57:
                        continue
                else :
                        return False

        return True

username = input("ENTER USERNAME - ")#username input taken
#2)-----
#username check is also happening at client why to bother server for it
#only alphabet - lower and upper and numberical value
while(is_correct(username)!=True):
        print("ERROR 100 Malformed username\n \n")
        username = input("ENTER USERNAME - ")


print("USERNAME IS VALID")
```

Figure 7:

2.3

If u are sending before registration instead of server giving u error client side is itself giving u error why to bother server for this. Also registration will be DONE in one step only u just need to just PRESS THE ENTER:)

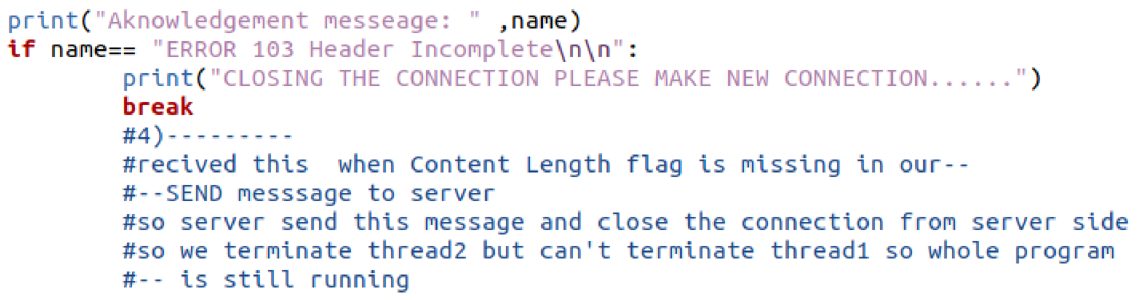


```
rtsm= "REGISTER TOSEND " + username + '\n\n'
rtrm ="REGISTER TORECV " + username + '\n\n'


love= input("Press ENTER TO REGISTER-")
while love!="":
        print("ERROR 101 No user registered \n \n")
        print("ERROR: REGISTER BEFORE SENDING MESSAGES")
        love= input("Press ENTER TO REGISTER-")

#3)---------------------
#at client only ERROR message of REGISTRATION IS NOT COMPLETED YET
#sender is not sending the the error message why to bother server for this
# this problem will be solved at client part only na..
```

Figure 8:

2.4

When in our message is Content Length field is missing server closes the connection but on client side there are two threads are running . After recieving this ERROR 103 Header Incomplete error reciving thread will end but our sending thread is still running so. Our whole client program will not terminate :(

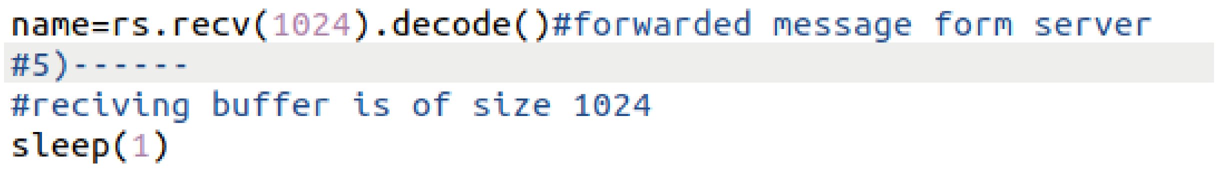


```
print("Aknowledgement messeage: " ,name)
if name== "ERROR 103 Header Incomplete\n\n":
        print("CLOSING THE CONNECTION PLEASE MAKE NEW CONNECTION......")
        break
        #4)---------
        #recived this  when Content Length flag is missing in our--
        #--SEND messsage to server
        #so server send this message and close the connection from server side
        #so we terminate thread2 but can't terminate thread1 so whole program
        #-- is still running
```

Figure 9:

2.5

Our receiving buffer will be of size 1024 everywhere



```
name=rs.recv(1024).decode()#forwarded message form server
#5)------
#reciving buffer is of size 1024
sleep(1)
```

Figure 10: