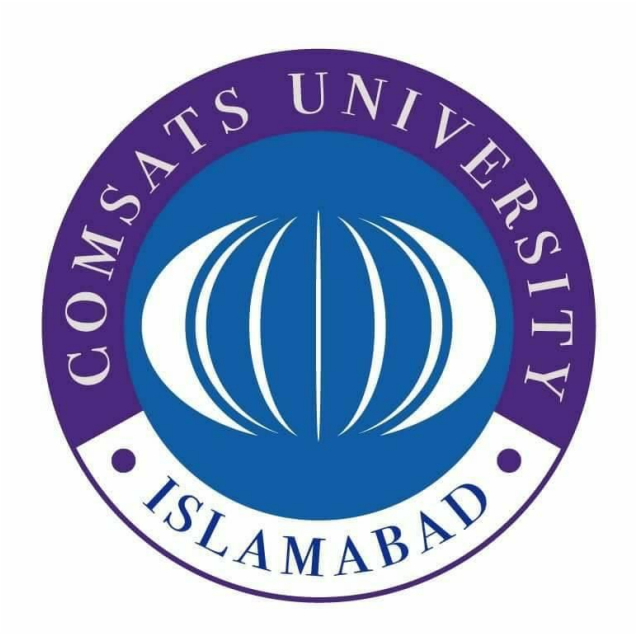
COMPUTER COMMUNICATION AND NETWORKING



SEMESTER PROJECT

MULTI THREADED SERVER AND CLIENTS WITH JEOPARDY PROTOCOL IMPLEMENTATION

Name	Hassan Ahmed, Muhammad Ali Buttar, M. Noman Ijaz
Roll No.	FA17-BSE-051, FA17-BSE-075, FA17-BSE-090
Class	BSE 5A
Submitted To	Dr. Munam Ali Shah

Contents

Abstract

This project is about implementation about a multi-threaded server and client. A server can accept more than one client. Server and Clients interact with each other to play Jeopardy game of questions and answers and client's respective scores are updated

Introduction

We developed a multi-threaded client/server application. The server we developed can handle more than one connection at a time. The main goal is to implement jeopardy game, where more than one client can connect to server and able to play the game.

There are two categories. When two or more than two clients are connected to the server then server chooses randomly one client and allow that client to choose one of the two categories.

When category is selected than server throws first question from the quiz from chosen category. One of the clients has to ring the buzzer and then have to answer the question. If answer is correct or answer is wrong the respected client's score is updated. After updating score, the first round will come to an end and then server starts second round.

After completing all rounds from all categories, the server will finish the game and notifies the clients who is the winner.

Design Details

Threading

Server Side

A pool of single thread is created which allotted to only accepting any incoming connections at any time after the socket has been created and binded. For every connection accepted there is a specific class for threading and clients in which every connection from a client creates a new connection and two threads are created and started for each client. One thread manages sending the messages to the client while the second thread manages receiving messages from the client. The connection object for each client is passed into the method which receives the messages so that each client can send messages on their own separate connection.

There are methods created for sending and receiving messages. Each of them contains the necessary functionality which deals with how messages from the Clients are interpreted and also how the messages sent from the client will notify the clients about starting the new round, notifying the connected clients that they are in the game etc.

Methods/ Classes in Server

1. ClientThread Class

```
class ClientThread(threading.Thread):


    def __init__(self, ip, port1, socket1, i, conn12):
        threading.Thread.__init__(self)
        self.ip = ip
        self.port1 = port1
        self.socket1 = socket1
        self.i = i
        self.conn12 = conn12
        print("[+] New thread started for " + self.ip + ":" + str(self.port1))

    def run(self):
        # print("Connection from : " + self.ip + ":" + str(self.port1))

        if self.i == 0:
            send_message()

        if self.i == 1:
            recv_message(self.conn12)
```

This class is responsible for initiating the connection object. The class basically accepts port number, IP of client.

The next method ‘run’ basically assign task to thread. The thread is sending messages from server as well as receive messages from clients.

2. Create Socket Method

```
def create_socket():
    try:
        global host
        global port
        global s

        s = socket.socket()
        host = ''
        port = 5000

        print("Socket Created")
    except socket.error as msg:
        print("error creating the socket " + str(msg))


def bind_socket():
    global host
    global port
    global s
    print("Socket binded to : " + str(port))
    s.bind((host, port))
    s.listen(5)  # number of clients that can be connected
    print("Socket is Listening")
```

As the names of both methods define that these methods are creating and binding socket respectively.

3. Accept Connection Method

```
def accept_connections():
    global host
    global port
    global s
    global conn
    for c in all_conn:
        c.close()
    del all_addr[:]
    del all_conn[:]
    while True:
        try:
            i = 0
            conn, address = s.accept()
            s.setblocking(1)
            newthread = ClientThread(host, port, s, i, conn)
            newthread.start()
            threads.append(newthread)
            i = 1
            newthread1 = ClientThread(host, port, s, i, conn)
            newthread1.start()
            recv_threads.append(newthread1)
            all_conn.append(conn)
            all_addr.append(address)
            # list_connections()
        except:
            print("error in accepting connections")
```

The method basically accepts the connections from clients respectively.

Client Side

Each client contains a pool of two threads as well which are used similar to how they are used in the server, each one is used for sending and receiving messages. After the necessary methods of creating, binding the socket, the request for connection is sent to the server. Each of the method for two threads is called upon to start the process of communicating with the server. All the Questioning, score etc. are handled by the server

1. Send Message Method

```
def send_message():
    while True:
        try:
            message = input("\nClient: ")
            if message == 'ring':
                s.send(str.encode(client, 'utf-8'))
            if message != 'ring':
                s.send(str.encode(message, 'utf-8'))
        except:
            print("error in sending ")
            break
```

This method is responsible for sending massage to server.

2. Receiving message method

```
def recv_message():
    while True:
        try:
            message = str(s.recv(20480), 'utf-8')
            print("\n From Server: " + message)
        except:
            print("error in receiving")
            break
```

This method is responsible for receiving massage from server.

Jeopardy Functionality

The two methods in Server called “Send_Message()” and “Receiveing_Message(conn)” handle all of the functionality of the jeopardy game functionality. The send message starts the new round and notifies the clients that are in the game and then randomly one of them is presented with the option to choose the category. The answer by the client is taken as input in the receiving message method and then all the clients are presented with the question. Whoever types ‘ring’ first will be able to answer the question. Again, the answer will be checked on server side if it is correct or wrong and then the new round will be started by the server.

Code

SERVER

```
import socket
import threading
from queue import Queue
from random import randrange

NUMBER_OF_THREAD = 1
JOBS = [1]
threads = []
recv_threads = []

all_conn = []
all_addr = []

queue = Queue()

conn = None

cat = ['Animals', 'Countries']

q1 = ['How many legs does cat have ?', 'how many legs does chicken have ?']
a1 = ['4', '2']
q2 = ['Whats the area of Pakistan ?', 'Name a superpower ?']
a2 = ['8', 'pak']

score1 = 0
score2 = 0
cat_id = 0
n = 0
m = 0
client = 0

class ClientThread(threading.Thread):


    def __init__(self, ip, port1, socket1, i, conn12):
        threading.Thread.__init__(self)
        self.ip = ip
        self.port1 = port1
        self.socket1 = socket1
        self.i = i
        self.conn12 = conn12
        print("[+] New thread started for " + self.ip + ":" + str(self.port1))

    def run(self):
        # print("Connection from : " + self.ip + ":" + str(self.port1))

        if self.i == 0:
            send_message()

        if self.i == 1:
            recv_message(self.conn12)
```

```
def create_socket():
    try:
        global host
        global port
        global s

        s = socket.socket()
        host = ''
        port = 5000

        print("Socket Created")
    except socket.error as msg:
        print("error creating the socket " + str(msg))


def bind_socket():
    global host
    global port
    global s

    print("Socket binded to : " + str(port))
    s.bind((host, port))
    s.listen(5)  # number of clients that can be connected
    print("Socket is Listening")


def accept_connections():
    global host
    global port
    global s
    global conn

    for c in all_conn:
        c.close()

    del all_addr[:]
    del all_conn[:]

    while True:
        try:
            i = 0
            conn, address = s.accept()
            s.setblocking(1)

            newthread = ClientThread(host, port, s, i, conn)
            newthread.start()
            threads.append(newthread)
            i = 1
            newthread1 = ClientThread(host, port, s, i, conn)
            newthread1.start()
            recv_threads.append(newthread1)

            all_conn.append(conn)
            all_addr.append(address)
```

```
            # list_connections()
        except:
            print("error in accepting connections")


def send_message():
    while True:
        try:
            msg = input("Server Message: ")
            if msg == 'start':
                in_game()
                new_round()
                get_target()
            if msg == 'quit':
                break;
            if msg == 'list':
                list_connections()
            if msg == 'finish':
                winner()
            if len(str.encode(msg)) > 0 and msg!='start':
                for x in all_conn:
                    x.send(str.encode(msg))
        except:
            print("error sending message")
            break


def get_target():
    conn1 = None
    try:
        rand = randrange(len(all_conn))
        conn1 = all_conn[rand]
        i = 0

        conn1.send(str.encode('Press 1 for '+cat[0] + '\n' + 'Press 2 for '+cat[1]
+'\n'))

        for x in all_conn:
            if x != conn1:
                x.send(str.encode('Please wait\n'))
            i += 1

    except:
        print("selection no valid")
        return None


def in_game():
    for x in all_conn:
        x.send(str.encode("You are in the game"))


def new_round():
    for x in all_conn:
```

```
        x.send(str.encode('The new Round has started !!\n '))


def send_all(message):
    for x in all_conn:
        x.send(str.encode(message))


def winner():
    global score1
    global score2

    if(score1 > score2):
        send_all('The winner of the game is client1 with score '+str(score1))
        send_all('The loser is client2 with score '+str(score2))
    else:
        send_all('The winner of the game is client2 with score ' + str(score2))
        send_all('The loser is client1 with score ' + str(score1))



def recv_message(conn1):
    global cat_id
    global n
    global m
    global client
    global score1
    global score2

    while True:
        try:
            if conn1 is not None:
                # receiving messages
                msg = str(conn1.recv(20480), 'utf-8')

                # who rang the bell
                if 'ring' in msg[2:]:
                    client = msg
                    client = client[0:1]
                    print("THIS IS " + client)
                    send_all('The buzzer has been rang by '+client)

                # check the answer conditions


                elif cat_id == 0 and msg == '1':
                    cat_id = 1
                    send_all('Category Animal has been selected')
                    send_all(q1[n])

                elif cat_id == 1 and msg == a1[n]:
                    send_all('client '+client+' has given correct answer\n')
                    cat_id = 0
                    n = n + 1
```

```
                    if client == '1':
                        score1 = score1 + 10
                    else:
                        score2 = score2 + 10
                    send_all('Client1 got ' + str(score1))
                    send_all('client2 got ' + str(score2)+'\n')
                    if (n > 1):
                        n = 0
                        send_all('Note: Catagory 1 Questions are completed. Reset to
0')

                elif cat_id == 1 and msg!= a1[n] and 'ring' not in msg:
                    send_all('client ' + client + ' has not given correct answer\n')
                    cat_id = 0
                    n = n + 1

                    send_all('Client1 got ' + str(score1) + '\n')
                    send_all('client2 got ' + str(score2) + '\n')

                    if (n > 1):
                        n = 0
                        send_all('Note: Catagory 1 Questions are completed. Reset to
0')

                elif cat_id == 0 and msg == '2':
                    cat_id = 2
                    send_all('Category Countries has been selected\n')
                    send_all(q2[m])

                elif cat_id == 2 and msg == a2[m]:
                    send_all('client '+client+' has given correct answer')
                    cat_id = 0
                    m = m + 1
                    if client == '1':
                        score1 = score1 + 10
                    else:
                        score2 = score2 + 10
                    send_all('Client1 got ' + str(score1) +'\n')
                    send_all('client2 got ' + str(score2) + '\n')

                    if (m > 1):
                        m = 0
                        send_all('Note: Catagory 2 Questions are completed. Reset to
0')

                elif cat_id == 2 and msg != a1[m] and 'ring' not in msg:
                    send_all('client ' + client + ' has not given correct answer\n')
                    cat_id = 0
                    m = m + 1
                    send_all('Client1 got ' + str(score1))
                    send_all('client2 got ' + str(score2) + '\n')
                    if (m > 1):
                        m = 0
                        send_all('Note: Catagory 2 Questions are completed. Reset to
0')
```

```
                print("Client Message: " + msg)
        except:
            print("error in receiving messages from client")
            break


def list_connections():
    print("----Clients----- \n ")
    for i, conn in enumerate(all_conn):
        try:
            conn.send(str.encode(' '))
            # conn.recv(201480)
        except:
            del all_addr[i]
            del all_addr[i]
            continue

        print(str(i) + "   " + str(all_addr[i][0]) + "    " + str(all_addr[i][1]))


def create_workers():
    for _ in range(NUMBER_OF_THREAD):
        t = threading.Thread(target=work)
        t.daemon = True
        t.start()


def work():
    while True:
        x = queue.get()

        if x == 1:
            create_socket()
            bind_socket()
            accept_connections()

            for t in threads:
                t.join()
            for th in recv_threads:
                th.join()

        queue.task_done()


def create_jobs():
    for x in JOBS:
        queue.put(x)

    queue.join()
```

```
create_workers()
create_jobs()
```

CLIENT

```
import socket
import threading
from queue import Queue

client = '1 ring'

NUMBER_OF_THREAD = 2
JOBS = [1, 2]

queue = Queue()

s = socket.socket()
host = '127.0.0.1'    #write host server IP
port = 5000

s.connect((host, port))


def send_message():
    while True:
        try:
            message = input("\nClient: ")
            if message == 'ring':
                s.send(str.encode(client, 'utf-8'))
            if message != 'ring':
                s.send(str.encode(message, 'utf-8'))
        except:
            print("error in sending ")
            break


def recv_message():
    while True:
        try:
            message = str(s.recv(20480), 'utf-8')
            print("\n From Server: " + message)
        except:
            print("error in receiving")
            break


def create_workers():
    for _ in range(NUMBER_OF_THREAD):
        t = threading.Thread(target=work)
        t.daemon = True
        t.start()


def work():
    while True:
```

```
        x = queue.get()

        if x == 1:
            send_message()
        if x == 2:
            recv_message()

        queue.task_done()


def create_jobs():
    for x in JOBS:
        queue.put(x)

    queue.join()


create_workers()
create_jobs()
```