Name :Rama Teja Repaka SUID:264100460

Internet programming Assignment-1

Question

Servers are supposed to run for a long time without stopping—therefore, they must be designed to provide good service no matter what their clients do. Examine the server examples (TCPEchoServer.java) and list anything you can think of that a client might do to cause it to give poor service to other clients. Suggest improvements to fix the problems that you find.

Answer:

Observations from TCPEchoServer

TCPEchoServer could not handle multiple requests at the same time because it does not have multithreading Implemented in its code. So it can serve only single Request at a time.If a client connects while another is already being serviced, the server will not echo the new client's data until it has finished with the current client, although the new client will be able to send data as soon as it connects. This type of server is known as an *iterative server*.

Below are the some of ways that one client can give poor service to other clients as well as possible solutions.

1)Flooding the Server with Requests from bad client

Server handles all the requests sent by the bad client after successful socket connection. So if bad client sends flood of requests server will be handling all the requests sent by this client which leads to resource exhaustion and finally unavailability of server. Other clients cannot communicate with server due to this denial of service attack performed by bad client.This is one of the way one client can provides poor service to the other clients.

Flooding Solution:
Implement **Multithreading**

New thread is spawned to handle each client connection. This approach is called Thread per client.multithreading at the server side leads to handle multiple requests at the same time. So clients no need to wait to send data to server.

By the above approach server serves multiple clients but let's take a case where client try to Flood the server with echo's then in order to prevent that flooding implemented below logic in server.

Each time when server creates a connection with client it creates new thread so in the run method I am counting no of requests received so if it receives more than 20 requests then closing the client socket connection this way i am blocking client to prevent flooding.

You can change the no of requests to any number depending on that socket connection will close.

```
 int i=0; // variable to count the requests

        while ((inputLine = in.readLine()) != null)
           {

            i++;

            if(i==20) // condition to close the socket connection when any one try to flood with
messages more than 20
              break;

            }

      clientSocket.close();
```

2) **Idle Bad client**

Let's take a case where client is connected to the server but not having any sort of communication with it. This will not allow other clients to connect to the server.

Solution: Multithreading with Socket Connection timeout

New thread is spawned to handle each client connection. This approach is called Thread per client.multithreading at the server side leads to handle multiple requests at the same time. So clients still can communicate even some idle connections are there.

By the above approach server serves multiple clients but server resources are used for no purpose so efficient solution is configure Socket Connection timeout both in client and server socket in multithreaded server code.

Socket connection timeout makes socket connection close if idle activity is happening on socket connection after specified time out value.

In the code timeout is specified like shown below.

clientSocket.setSoTimeout(10000); // after connecting to client if it is idle and not sending any data above step closes the socket connection.

serverSocket.setSoTimeout(10000);

Implemented this in server while loop which keeps on listening if no one is connecting then closing the socket connection.

You might be wondering how if closed how it will handle

This step is written in while loop so

server socket is created and and check for clients if not found closes the connection

This process keeps on repeating.

Note :

started server with port 10008 so connecting Both bad clients to that port.

Execution Instructions:

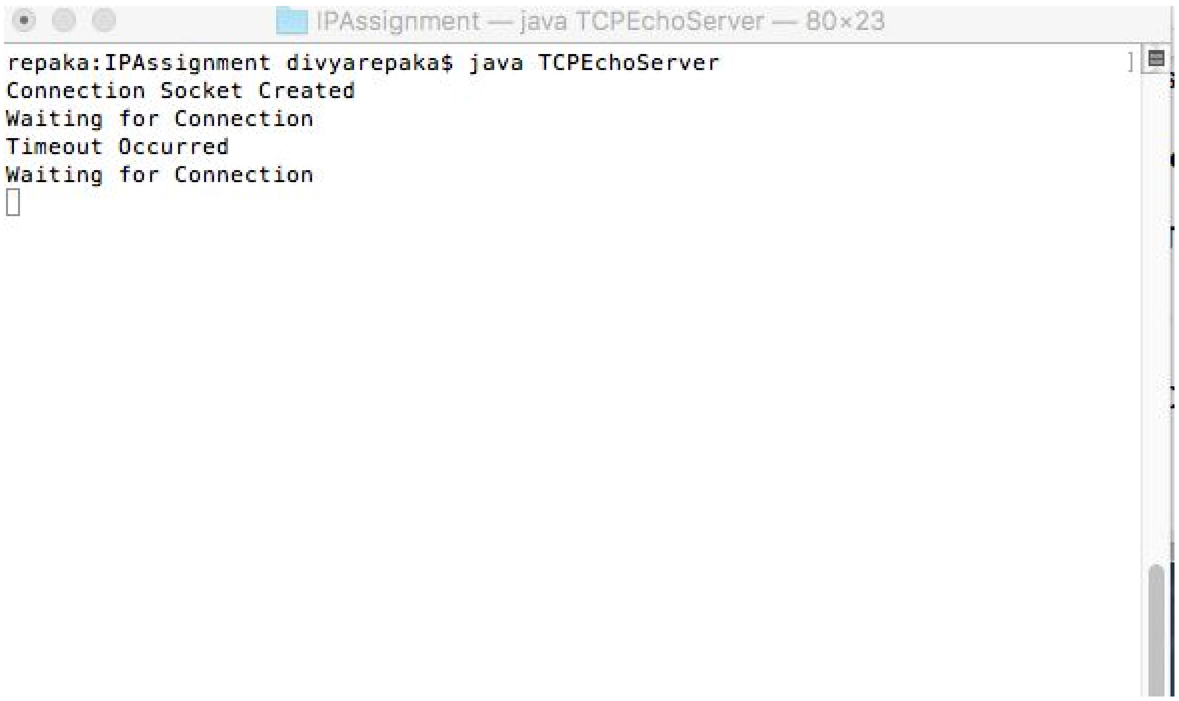
1) **To test flooding prevention**

Compile TCPEchoServer

javac TCPEchoServer.java

Run
java TCPEchoServer

Output



```
repaka:IPAssignment divyarepaka$ java TCPEchoServer
Connection Socket Created
Waiting for Connection
Timeout Occurred
Waiting for Connection
```

Server started successfully.

Compile TCPEchoClient1

javac TCPEchoClient1.java

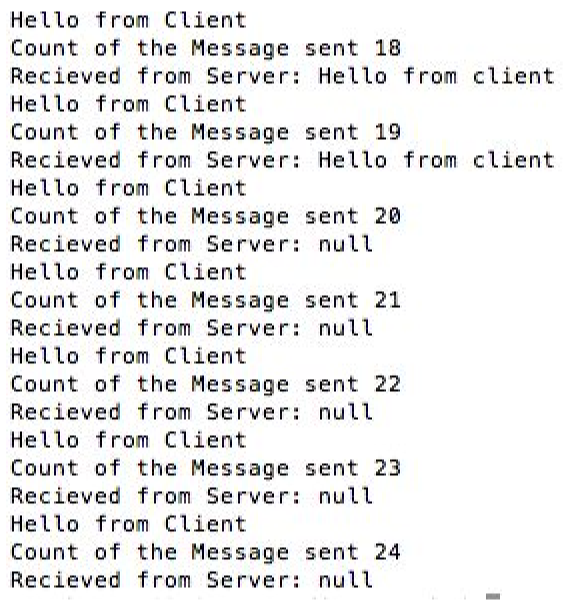
Run

java TCPEchoClient1

Output screenshot

Client side

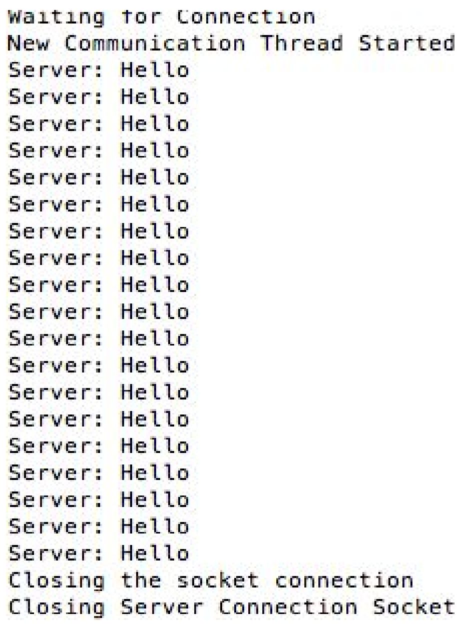
```
Attemping to connect to host 127.0.0.1 on port 10007.
Flooding Server with  Message: Hello from Client
Count of the Message sent 1
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 2
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 3
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 4
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 5
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 6
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 7
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 8
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 9
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 10
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 11
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 12
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 13
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 14
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 15
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 16
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 17
Recieved from Server: Hello from client
```



```
Hello from Client
Count of the Message sent 18
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 19
Recieved from Server: Hello from client
Hello from Client
Count of the Message sent 20
Recieved from Server: null
Hello from Client
Count of the Message sent 21
Recieved from Server: null
Hello from Client
Count of the Message sent 22
Recieved from Server: null
Hello from Client
Count of the Message sent 23
Recieved from Server: null
Hello from Client
Count of the Message sent 24
Recieved from Server: null
```

You can see server only responds for 20 messages after server closes the socket connection.

Server side screen shot:



```
Waiting for Connection
New Communication Thread Started
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Server: Hello
Closing the socket connection
Closing Server Connection Socket
```

2) To Test Idle client

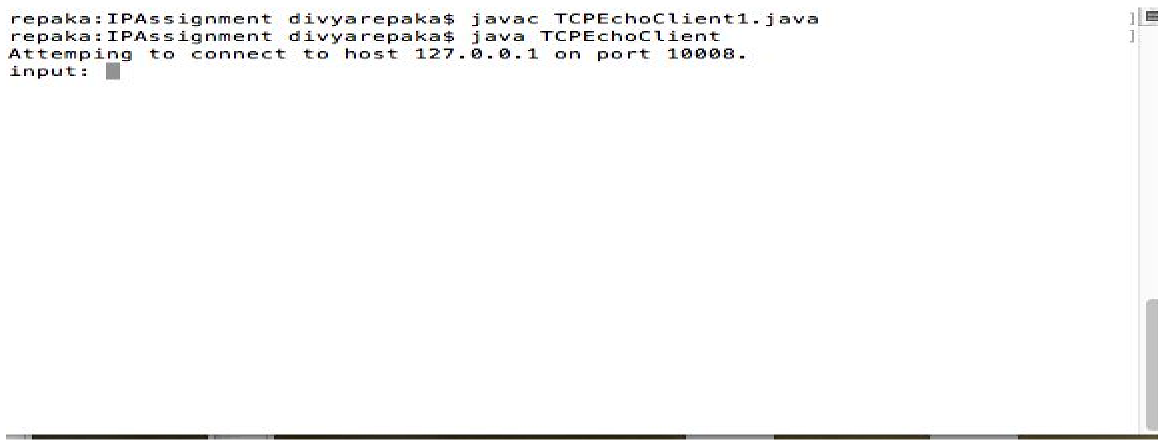
Compile TCPEchoClient

javac TCPEchoClient.java

Run

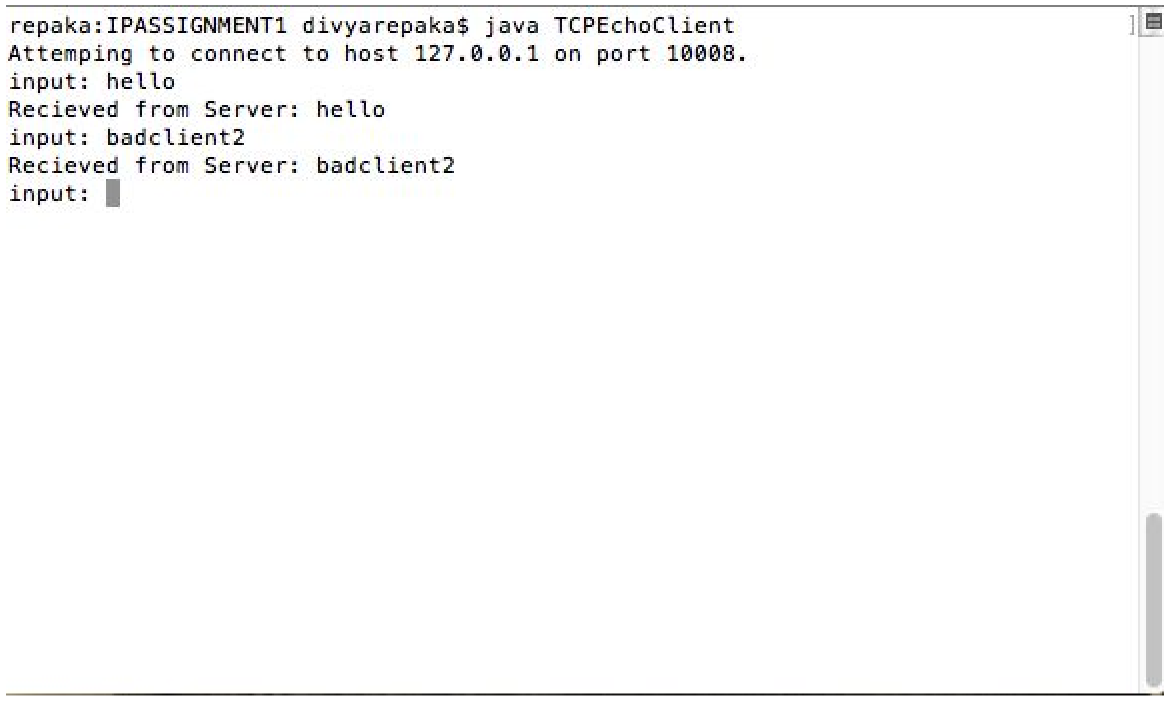
java TCPEchoClient

After connecting



```
repaka:IPAssignment divyarepaka$ javac TCPEchoClient1.java
repaka:IPAssignment divyarepaka$ java TCPEchoClient
Attemping to connect to host 127.0.0.1 on port 10008.
input: 
```

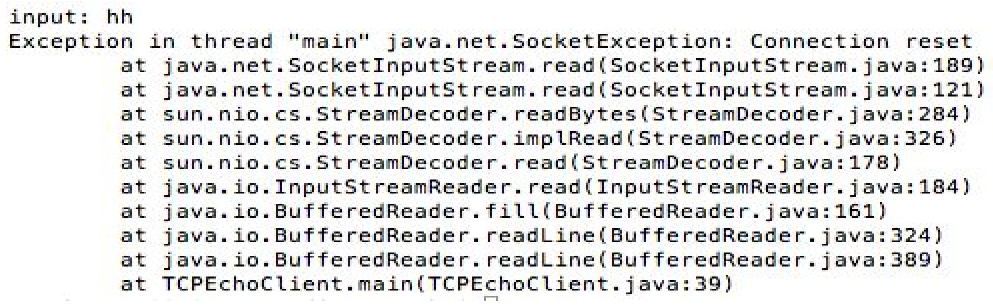
Entered some input and receive response from server



```
repaka:IPASSIGNMENT1 divyarepaka$ java TCPEchoClient
Attemping to connect to host 127.0.0.1 on port 10008.
input: hello
Recieved from Server: hello
input: badclient2
Recieved from Server: badclient2
input: 
```

Now be idle for some time and try to enter anything

connection reset Exception will occur at client side

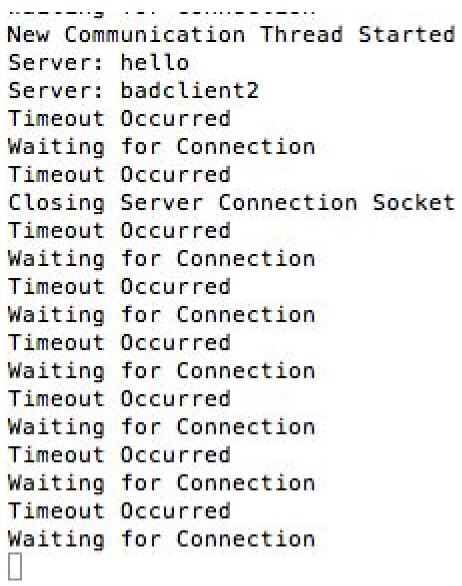


```
input: hh
Exception in thread "main" java.net.SocketException: Connection reset
        at java.net.SocketInputStream.read(SocketInputStream.java:189)
        at java.net.SocketInputStream.read(SocketInputStream.java:121)
        at sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:284)
        at sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:326)
        at sun.nio.cs.StreamDecoder.read(StreamDecoder.java:178)
        at java.io.InputStreamReader.read(InputStreamReader.java:184)
        at java.io.BufferedReader.fill(BufferedReader.java:161)
        at java.io.BufferedReader.readLine(BufferedReader.java:324)
        at java.io.BufferedReader.readLine(BufferedReader.java:389)
        at TCPEchoClient.main(TCPEchoClient.java:39)
```

Server side

At server side you can clearly see

Closing server connection socket which means client socket connection is closed at server side



```
New Communication Thread Started
Server: hello
Server: badclient2
Timeout Occurred
Waiting for Connection
Timeout Occurred
Closing Server Connection Socket
Timeout Occurred
Waiting for Connection
Timeout Occurred
Waiting for Connection
Timeout Occurred
Waiting for Connection
Timeout Occurred
Waiting for Connection
Timeout Occurred
Waiting for Connection
Timeout Occurred
Waiting for Connection
```

So by this screenshot we can clearly observe what’s happening

This clearly demonstrates both the requirements.