# هدف آزمایش

در این آزمایش قصد داریم تا شما را با برنامه­نویسی سوکت از طریق پیاده‌سازی یک نرم‌افزار پروکسی آشنا کنیم. روند آزمایش به این ترتیب خواهد بود که ابتدا مروری بر ساخت سوکت و مفاهیم مرتبط خواهیم داشت و در نهایت با بررسی پیاده‌سازی نرم‌افزار پروکسی آزمایشگاه را به پایان می‌بریم. آنچه در این جلسه می‌آموزید مقدمه‌ای است بر پروژه برنامه‌نویسی که انتهای این آزمایش برای شما تعریف خواهد شد.

# مقدمه

امروزه استفاده از پروکسی‌ها در لایه‌های مختلف اجتناب ناپذیر شده است. به‌عنوان مثال همان‌طور که در درس خواندید، پروتکل UDP فاقد مکانیسم‌های کنترل ازدحام است و می‌تواند باعث شود تا ارسال کنندگان داده با ارسال زیاد بسته‌های UDP باعث کاهش کارایی سیستم‌هایی شوند که از پروتکل TCP به عنوان پروتکل لایه انتقال خود استفاده می‌کنند. به همین علت ممکن است در برخی از شبکه‌ها استفاده از پروتکل UDP محدود شود. به این منظور کاربران این شبکه‌ها از پروکسی‌های لایه حمل استفاده می‌کنند که بسته‌های UDP را به بسته‌های TCP تبدیل کرده و پس از دریافت جواب این تبدیل معکوس را مجددا انجام می‌دهد.

# فعالیت­های قبل آزمایش

* مرور پروتکل­های لایه ارتباط
* مرور زبان برنامه نویسی جاوا
* داشتن یک محیط توسعه جاوا (ترجیحا IntelliJ IDE)

# شرح آزمایش

در ابتدا با یک مثال ساده سراغ برنامه­نویسی سوکت رفته و سپس قسمت اصلی آزمایش را که در ادامه توضیح داده می­شود باهم ادامه می­دهیم. توجه داشته باشد این دستورکار براساس نسخه‌ی ۸ جاوا نگارش شده است و در آن از کتابخانه‌های NIO و ... استفاده نشده است.

## برنامه نویسی سوکت در جاوا

1. ابتدا یک پروژه ساده جاوایی به نام SocketServer ایجاد می­کنیم در ذاخل این پروژه یک فایل به نام Server.java ایجاد می­کنیم و کد زیر را در داخل آن می­نویسیم:

import java.io.IOException;

import java.net.ServerSocket;

public class Server {

    ServerSocket mServer;

    int serverPort = 9090;

    public Server() {

        try {

            // create a server socket

            mServer = new ServerSocket(serverPort);

            System.out.println("Server Created!");

            // wait for client

            mServer.accept();

            // horaaaaa, the client is coming

            System.out.println("Connected to New Client!");

        } catch (IOException e) {

        }

}

    public static void main(String[] args) {

        new Server();

    }

}

برنامه Server.java را اجرا می کنیم ، مشاهده می کنیم که پیغام Server Created در خروجی چاپ می شود ولی پیغام Connected to New Client چاپ نمی شود ، دلیل این موضوع متد accept است ، این متد مانند یک حلقه انتظار است که آنقدر صبر می کند تا یک کلاینت درخواست اتصال به سرور را بدهد ، پس از اتصال کلاینت به سرور (و چند کار دیگر که در ادامه می بینیم) کار متد accept تمام می شود و کنترل برنامه به خط بعدی می رود.

به جز متد accept نکته خاص دیگری در کد بالا وجود ندارد ، برای توضیحات بیشتر می توان به شماره پورت اشاره کرد که شماره پورتی است که برنامه سرور بر روی آن منتظر اتصال کلاینت ها می شود. و اینکه برای ساخت سوکت سروری از کلاس ServerSocket استفاده می کنیم.

1. **ساخت و اجرای کلاینت:**

حال باید یک برنامه کلاینت بسازیم و به سروری قبلی متصل شویم ، دقت کنید که برنامه سرور قبلی همچنان در حال اجراست (زیرا منتظر کلاینت است).

یک پروژه جاوایی جدید به نام SocketClient ایجاد کنیم.در داخل این پروژه کلاس جدیدی به نام Client در داخل فایلی به نام Client.java ایجاد می کنیم و کد زیر را در آن می نویسیم :

import java.io.IOException;

import java.net.Socket;

import java.net.UnknownHostException;

public class Client {

    Socket mSocket;

    int port = 9090;

    String serverAddress = "127.0.0.1";

    public Client() {

        try {

            mSocket = new Socket(serverAddress, port);

            System.out.println("connect to server ....");

        } catch (UnknownHostException e) {

        } catch (IOException e) {

            System.out.println(e.getMessage());

        }

    }

    public static void main(String[] args) {

        new Client();

    }

}

برنامه فوق به شدت ساده است ، تنها کاری که کردیم با داشتن اطلاعات سرور (آی پی و شماره پورت) یک شی جدید از کلاس Socket ایجاد کردیم ، به محض ساخته شدن شی از این کلاس ، کلاینت سعی در برقراری ارتباط با سرور می کند.

برنامه Clinet.java را اجرا می کنیم.

پس از اینکه پیغام connect to server را در خروجی کلاینت مشاهده کردیم مجدداً به سراغ خروجی برنامه سرور می رویم ، می بینیم که خروجی زیر چاپ شده و برنامه سرور خاتمه یافته است.

Server Created!

Connected to New Client!

**نکته:** معمولاً برنامه های سرور و کلاینت دو برنامه متفاوت و جدا هستند ، برای همین دو پروژه جدا برای هر کدام ایجاد کردیم. در بسیاری از کاربرد ها معقول به نظر نمی رسد که در کنار کد های کلاینت کد های سرور را نیز داشته باشید. (به جز مواقع خاص که یک برنامه می تواند به انتخاب کاربر سرور یا کلاینت باشد ، به عنوان مثال در بازی هایی همچون Counter Strike شما قابلیت ایجاد سرور و کلاینت را همزمان در برنامه دارید.)

**نکته** : در مثال بالا ابتدا سرور ایجاد شد ، منتظر کلاینت ماند ، کلاینت با داشتن اطلاعات سرور یک شی از نوع Socket ایجاد کرد و به سرور متصل شد ، پس از اتصال کلاینت به سرور کار متد accept در برنامه سرور تمام شد ، خروجی مورد نظر چاپ شد و کار هر دو برنامه به پایان رسید ولی در کاربرد های واقعی معمولاً اینچنین نیست و سرور یک برنامه همیشه در حال اجراست ، کلاینت ها می آیند و می روند ولی سرور همیشه پا برجاست.

1. تبادل اطلاعات

در قسمت قبل فقط کلاینت به سرور وصل شد و پس از آن هر دو برنامه خاتمه یافتند ولی فقط اتصال کافی نیست ، در یک برنامه تحت شبکه ما دوست داریم قادر به ارسال و دریافت داده و اطلاعات باشیم (چه بدیهی:دی) ، در ادامه سعی می کنیم یک مکالمه بسیار ساده بین سرور و کلاینت داشته باشیم تا با نحوه ارسال اطلاعات آشنا شوید. برای ادامه فرض می‌کنیم با کتابخانه‌های io و streamها در جاوا آشنایی دارید.

کد Server را به صورت زیر تغییر می دهیم :

import java.io.DataInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class Server {

    ServerSocket mServer;

    int serverPort = 9090;

    InputStream fromClientStream;

    OutputStream toClientStream;

    DataInputStream reader;

    PrintWriter writer;

    public Server() {

        try {

            // create server socket!

            mServer = new ServerSocket(serverPort);

            System.out.println("Server Created!");

            // wait for client

            // hold an object of Socket for each client

            Socket client = mServer.accept();

            // horaaaaa

            System.out.println("Connected to New Client!");

            // input stream (stream from client)

            fromClientStream = client.getInputStream();

            // output sream (stream to client)

            toClientStream = client.getOutputStream();

            reader = new DataInputStream(fromClientStream);

            writer = new PrintWriter(toClientStream, true);

            // send message to client

            writer.println("Salam Client joon");

            System.out.println("Server :Salam Client joon");

            // Receive client response (javab:D)

            String javab = reader.readLine();

            System.out.println("Client :" + javab);

                // send message to client (again)

            writer.println("khobi??");

            System.out.println("Server :khobi?");

            // Receive client response (javab:D)

            javab = reader.readLine();

            System.out.println("Client :" + javab);

        } catch (IOException e) {

        }

    }

    public static void main(String[] args) {

        new Server();

    }

}

کد client را به شکل زیر تغییر می­دهیم:

import java.io.DataInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.io.PrintWriter;

import java.net.Socket;

import java.net.UnknownHostException;

public class Client {

    Socket mSocket;

    int port = 9090;

    String serverAddress = "127.0.0.1";

    InputStream fromServerStream;

    OutputStream toServerStream;

    DataInputStream reader;

    PrintWriter writer;

    public Client() {

        try {

            mSocket = new Socket(serverAddress, port);

            System.out.println("connect to server ....");

            // input stream (stream from server)

            fromServerStream = mSocket.getInputStream();

            // output stream (stream to server)

            toServerStream = mSocket.getOutputStream();

            reader = new DataInputStream(fromServerStream);

            writer = new PrintWriter(toServerStream, true);

            // first : read server message

            String msg = reader.readLine();

            System.out.println("Server :" + msg);

            // send message to server

            writer.println("Salam Servere man");

            System.out.println("Client :Salam Servere man");

            // read server message

            msg = reader.readLine();

            System.out.println("Server :" + msg);

            // send message to server

            writer.println("Ohum!!!");

            System.out.println("Client :Ohum!!!");

        } catch (UnknownHostException e) {

        } catch (IOException e) {

            System.out.println(e.getMessage());

        }

    }

    public static void main(String[] args) {

        new Client();

    }

}

مانند حالت قبل، ابتدا برنامه سرور و سپس برنامه کلاینت را اجرا می کنیم.

خروجی برنامه سرور :

Server Created!

Connected to New Client!

Server :Salam Client joon

Client :Salam Servere man

Server :khobi?

Client :Ohum!!!

خروجی برنامه کلاینت :

connect to server ....

Server :Salam Client joon

Client :Salam Servere man

Server :khobi??

Client :Ohum!!!

چند بار برنامه سرور و کلاینت را اجرا کنید و سعی کنید با کمک مدرس خود، رفتار دو برنامه را درک و تحلیل کنید.

## پروکسی

این آزمایش از دو بخش تشکیل شده است: یک بخش برنامه پروکسی و بخش دیگر ابزاری است که به عنوان سرویس گیرنده[[1]](#footnote-1) به پروکسی متصل می‌شود. برای بخش سرویس دهنده از سرورهای واقعی موجود در اینترنت استفاده می‌شود و نیازی به پیاده سازی نیست. برنامه پروکسی بین سرویس گیرنده و سرویس دهنده قرار دارد و درخواست‌های سرویس گیرنده را به قالب مناسب تبدیل کرده و برای سرویس دهنده ارسال می‌کند.

برنامه پروکسی قادر است، پیغام‌های UDP ارسال شده از طرف سرویس گیرنده را بر روی اتصال TCP به سمت سرویس دهنده که بر روی پورت مشخصی و در اینترنت در حال گوش کردن است ارسال کند. جواب درخواست‌ها بر روی اتصال UDP به سرویس گیرنده ارسال می‌شود.

### سرویس گیرنده

همان‌گونه که مشخص است برنامه‌های نوشته شده فعلی قادر به کارکردن با پروکسی فوق نیستند. برای همین ابزاری مناسب این پروکسی باید ایجاد شود. این ابزار از کاربرد HTTP بر روی پروتکل UDP اینترنت پشتیبانی می‌کند.

### کاربرد HTTP بر روی پروتکل UDP

برنامه مشابه یک مرورگر اینترنت، البته با قابلیت‌های بسیار کمتر عمل می‌کند. برنامه به عنوان ورودی یک درخواست HTTP استاندارد را از کاربر دریافت کرده و آن را در قالب بسته UDP به پروکسی ارسال کند. برای سادگی صرفا برنامه قادر به پشتیبانی از GET است. مثالی از ورودی به برنامه در زیر آمده است:

GET / HTTP/1.1

Host: aut.ac.ir

در این حالت برنامه بسته UDP را ایجاد کرده و آن را به پروکسی که بر روی پورت UDP مشخصی تنظیم شده است ارسال می‌کند. پروکسی مطابق با درخواست دریافت شده، یک ارتباط TCP با میزبان با نام دامنه aut.ac.ir بر روی پورت پیش‌فرض 80 برقرار می‌کند. سپس پروکسی درخواست HTTP را به وب سرور مورد نظر در پورت 80 ارسال می‌کند و منتظر جواب می‌ماند. پس از دریافت کامل جواب، بسته یا بسته‌های UDP را ایجاد کرده و جواب را با استفاده از این بسته‌ها به سمت سرویس گیرنده ارسال می‌کند. برنامه پس از دریافت کامل بسته‌ها، یک فایل HTML ایجاد کرده و محتویات بسته‌ها را در آن قرار می‌دهد.

## پاسخ دهید

* از نظر شما پراکسی چگونه متوجه می‌شود که مقصد ارتباط کجاست؟
* با توجه به اینکه ارتباط TCP یک ارتباط قابل اطمینان و UDP یک ارتباط غیرقابل اطمینان می‌باشد، کاربر نهایی (شخصی که با سرویس‌گیرنده در ارتباط می‌باشد) ارتباط را چگونه خواهد دید؟ قابل اطمینان یا غیرقابل اطمینان؟

## انجام دهید

قسمتی از کد این پراکسی از پیش تهیه شده است. این قسمت شامل یک پراکسی ساده بوده که پیام‌های HTTP را از طریق UDP دریافت کرده، آن‌ها را پردازش می‌کند. در این پردازش مشخص می‌شود مقصد ارتباط موردنظر کجا بوده و در گام بعدی یک ارتباط TCP با آن برقرار می‌گردد. تقاضایی که پیشتر بر روی UDP دریافت شده بود بر روی TCP به این مقصد ارسال شده و پاسخ آن دوباره به همان پورت و IP بسته‌ی UDP بازارسال می‌گردد.

* قسمت binding این کد تنها بر روی localhost می‌باشد، آن را به نحوی تغییر دهید که:

1. بر روی تمام کارت‌های شبکه bind شود.
2. بر روی کارت شبکه‌ای با یک آدرس IP مشخص bind شود.

* قسمت پردازش این کد را تکمیل کنید. (راهنمایی: از پاسخ سوال اول قسمت پاسخ‌ دهید، برای این قسمت استفاده کنید.)

## پروژه‌ی پایانی

صورت پروژه در قالب سند دیگری در اختیار شما قرار خواهد گرفت و زمان تحویل آن به شما اطلاع داده خواهد شد. این پروژه بر اساس قسمت پراکسی همین دستور کار می‌باشد و شما می‌بایست مواردی که در آن پروژه از شما خواسته شده است را به کد این آزمایش اضافه کنید. بدیهی است که کد شما می‌بایست به زبان جاوا بوده و تنها با استفاده از کتابخانه‌های استاندارد جاوا نوشته شود.

**موفق باشید.**

1. client [↑](#footnote-ref-1)