**Universitatea Tehnică a Moldovei**

**Lucrare de laborator 1**

**LA DISCIPLINA:** PROGRAMAREA IN RETEA

**CU TEMA:** Socket | HTTP | Multithreading

**Student:**

Lupei Nicolae FI-171

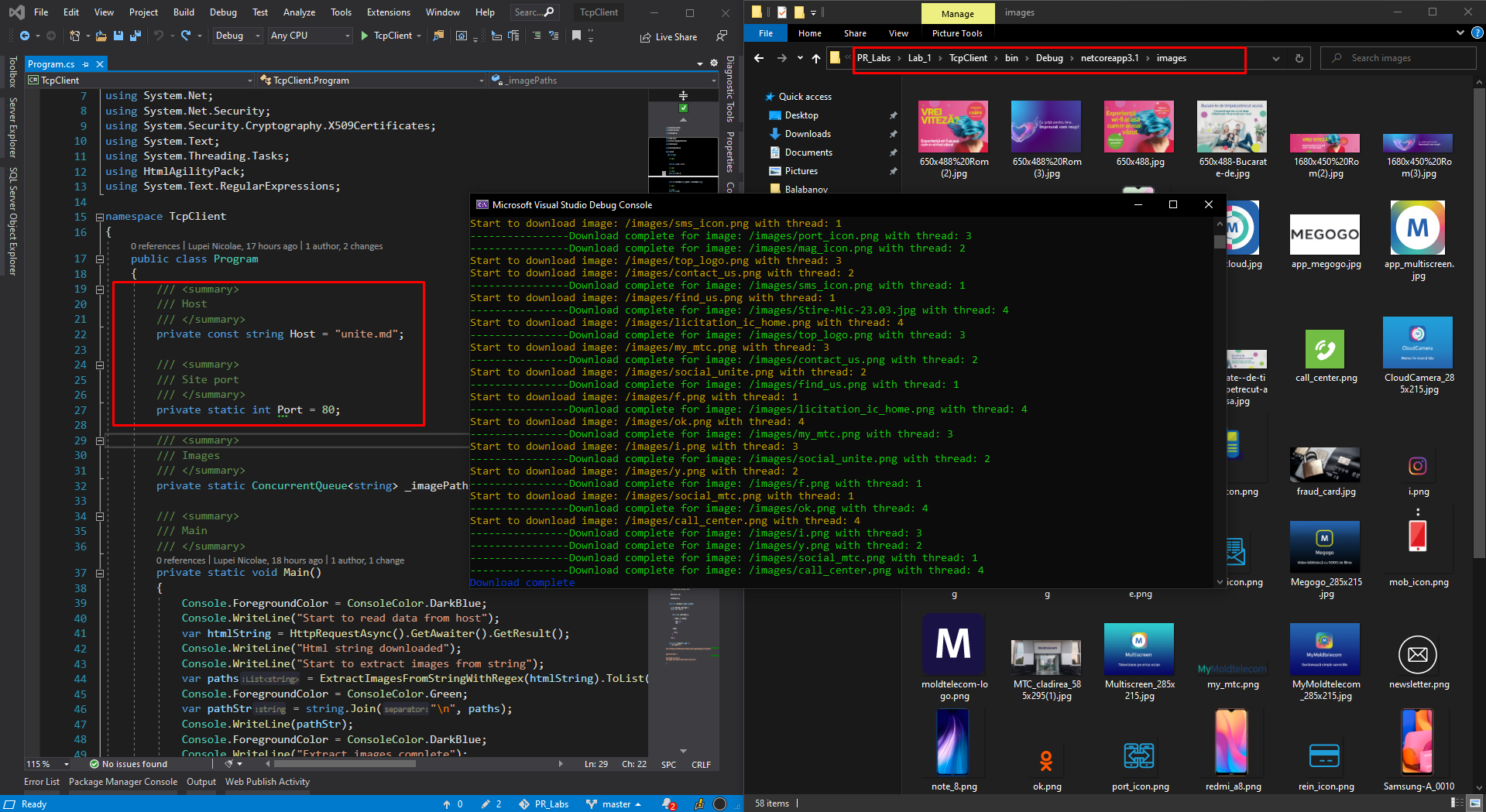
**Profesor:**

Boldumac Oleg

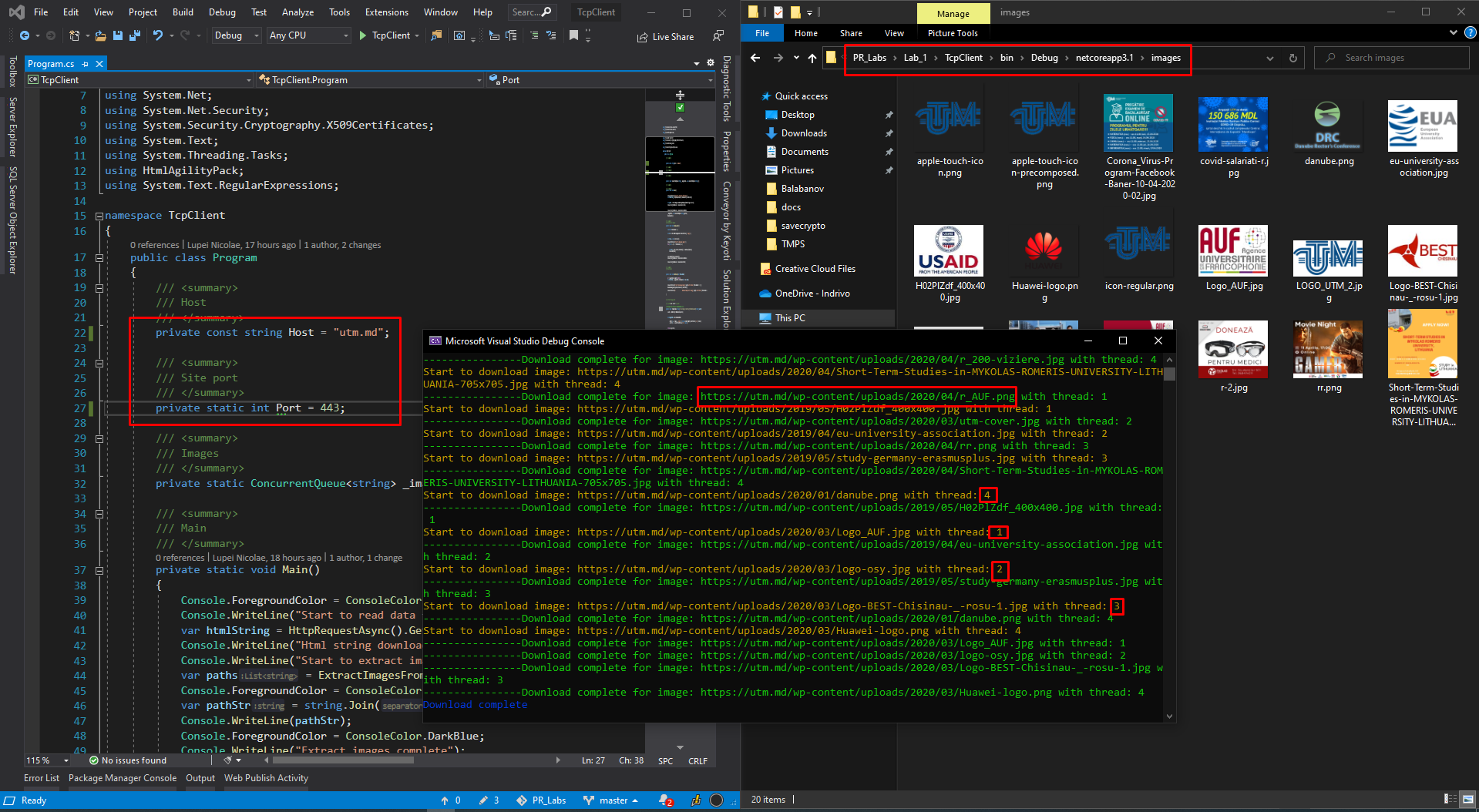
Chișinău – 2020

**Mersul lucrarii:**

Program rulat pentru unite.md



Program rulat pentru utm.md



**Codul sursa:**

using System;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.IO;

using System.IO.Compression;

using System.Linq;

using System.Net;

using System.Net.Security;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using HtmlAgilityPack;

using System.Text.RegularExpressions;

namespace TcpClient

{

    public class Program

    {

        /// <summary>

        /// Host

        /// </summary>

        private const string Host = "unite.md";

        /// <summary>

        /// Site port

        /// </summary>

        private static int Port = 80;

        /// <summary>

        /// Images

        /// </summary>

        private static ConcurrentQueue<string> \_imagePaths = new ConcurrentQueue<string>();

        /// <summary>

        /// Main

        /// </summary>

        private static void Main()

        {

            Console.ForegroundColor = ConsoleColor.DarkBlue;

            Console.WriteLine("Start to read data from host");

            var htmlString = HttpRequestAsync().GetAwaiter().GetResult();

            Console.WriteLine("Html string downloaded");

            Console.WriteLine("Start to extract images from string");

            var paths = ExtractImagesFromStringWithRegex(htmlString).ToList();

            Console.ForegroundColor = ConsoleColor.Green;

            var pathStr = string.Join("\n", paths);

            Console.WriteLine(pathStr);

            Console.ForegroundColor = ConsoleColor.DarkBlue;

            Console.WriteLine("Extract images complete");

            \_imagePaths = new ConcurrentQueue<string>(paths);

            DownloadAll();

        }

        /// <summary>

        /// Download all images

        /// </summary>

        private static void DownloadAll()

        {

            const int threadCount = 4;

            var dir = Path.Combine(AppContext.BaseDirectory, "images");

            if (Directory.Exists(dir)) Directory.Delete(dir, true);

            var tasks = new List<Task>();

            Console.WriteLine("Start to download images:");

            for (var i = 0; i < threadCount; i++)

            {

                var index = i;

                var task = Task.Factory.StartNew(() => RunThread(index));

                tasks.Add(task);

            }

            Task.WaitAll(tasks.ToArray());

            Console.ForegroundColor = ConsoleColor.DarkBlue;

            Console.WriteLine("Download complete");

            Console.ForegroundColor = ConsoleColor.White;

        }

        /// <summary>

        /// Run thread

        /// </summary>

        /// <param name="threadIndex"></param>

        private static void RunThread(int threadIndex)

        {

            if (\_imagePaths.IsEmpty) return;

            var extracted = \_imagePaths.TryDequeue(out var path);

            if (!extracted) return;

            Console.ForegroundColor = ConsoleColor.DarkYellow;

            Console.WriteLine($"Start to download image: {path} with thread: {threadIndex + 1}");

            DownloadImageAsync(path).GetAwaiter().GetResult();

            Console.ForegroundColor = ConsoleColor.Green;

            Console.WriteLine($"----------------Download complete for image: {path} with thread: {threadIndex + 1}");

            RunThread(threadIndex);

        }

        /// <summary>

        /// Extract image paths

        /// </summary>

        /// <param name="source"></param>

        /// <returns></returns>

        // ReSharper disable once UnusedMember.Local

        private static IEnumerable<string> ExtractImagesFromStringWithHtmlDocument(string source)

        {

            source = WebUtility.HtmlDecode(source);

            var doc = new HtmlDocument();

            doc.LoadHtml(source);

            var imagePaths = doc.DocumentNode

                .Descendants()

                .Where(x => x.Name == "img" && !string.IsNullOrEmpty(x.Attributes["src"].Value))

                .Select(x => x.GetAttributeValue("src", null))

                .ToList();

            return imagePaths;

        }

        /// <summary>

        /// Extract image paths

        /// </summary>

        /// <param name="source"></param>

        /// <returns></returns>

        private static IEnumerable<string> ExtractImagesFromStringWithRegex(string source)

        {

            //const string expr = "<img.\*?src=\"(.\*?)\"[^\\>]+>";

            //var result = Regex.Matches(source, expr);

            //var images = result.Select(x => x.Groups[1].Value)

            //    .Where(x => !string.IsNullOrEmpty(x))

            //    .Distinct()

            //    .ToList();

            //return images;

            const string expr = "(?<=\\b=\")[^\"]\*";

            var result = Regex.Matches(source, expr);

            var images = result.Select(x => x.Groups[0].Value)

                .Where(x => !string.IsNullOrEmpty(x) &&

                            (x.EndsWith(".png") || x.EndsWith(".jpg"))

                            && (!x.StartsWith("http") || x.Contains(Host)))

                .Distinct()

                .ToList();

            return images;

        }

        /// <summary>

        /// Http request

        /// </summary>

        /// <returns></returns>

        private static async Task<string> HttpRequestAsync()

        {

            using var tcp = new System.Net.Sockets.TcpClient(Host, Port);

            Stream networkStream = tcp.GetStream();

            tcp.SendTimeout = 500;

            tcp.ReceiveTimeout = 1000;

            // Send request headers

            var builder = new StringBuilder();

            builder.AppendLine(GetHeaders());

            builder.AppendLine();

            var requestHeaders = builder.ToString();

            var header = Encoding.ASCII.GetBytes(requestHeaders);

            if (Port == 443)

            {

                var sslStream = new SslStream(networkStream, false, ValidateServerCertificate, null);

                await sslStream.AuthenticateAsClientAsync("127.0.0.1");

                networkStream = sslStream;

            }

            await networkStream.WriteAsync(header, 0, header.Length);

            // receive data

            string result;

            await using (var memory = new MemoryStream())

            {

                await networkStream.CopyToAsync(memory);

                memory.Position = 0;

                var data = memory.ToArray();

                var index = BinaryMatch(data, Encoding.ASCII.GetBytes("\r\n\r\n")) + 4;

                var headers = Encoding.ASCII.GetString(data, 0, index);

                memory.Position = index;

                if (headers.IndexOf("Content-Encoding: gzip", StringComparison.Ordinal) > 0)

                {

                    await using var decompressionStream = new GZipStream(memory, CompressionMode.Decompress);

                    await using var decompressedMemory = new MemoryStream();

                    decompressionStream.CopyTo(decompressedMemory);

                    decompressedMemory.Position = 0;

                    result = Encoding.UTF8.GetString(decompressedMemory.ToArray());

                }

                else

                {

                    result = Encoding.UTF8.GetString(data, index, data.Length - index);

                    //result = Encoding.GetEncoding("gbk").GetString(data, index, data.Length - index);

                }

            }

            networkStream.Close();

            return result;

        }

        /// <summary>

        /// Validate server

        /// </summary>

        /// <param name="sender"></param>

        /// <param name="certificate"></param>

        /// <param name="chain"></param>

        /// <param name="sslPolicyErrors"></param>

        /// <returns></returns>

        public static bool ValidateServerCertificate(object sender, X509Certificate certificate,

            X509Chain chain, SslPolicyErrors sslPolicyErrors)

        {

            return true;

        }

        /// <summary>

        /// Download image

        /// </summary>

        /// <param name="imagePath"></param>

        /// <returns></returns>

        private static async Task DownloadImageAsync(string imagePath)

        {

            using var tcp = new System.Net.Sockets.TcpClient(Host, Port);

            Stream networkStream = tcp.GetStream();

            tcp.SendTimeout = 500;

            tcp.ReceiveTimeout = 1000;

            // Send request headers

            var builder = new StringBuilder();

            builder.AppendLine(GetHeaders(imagePath));

            builder.AppendLine();

            var header = Encoding.ASCII.GetBytes(builder.ToString());

            if (Port == 443)

            {

                var sslStream = new SslStream(networkStream, false, ValidateServerCertificate, null);

                await sslStream.AuthenticateAsClientAsync("127.0.0.1");

                networkStream = sslStream;

            }

            await networkStream.WriteAsync(header, 0, header.Length);

            // receive data

            var memory = new MemoryStream();

            await networkStream.CopyToAsync(memory);

            memory.Position = 0;

            var data = memory.ToArray();

            var index = BinaryMatch(data, Encoding.ASCII.GetBytes("\r\n\r\n")) + 4;

            var imgName = imagePath.Split("/").LastOrDefault();

            var dir = Path.Combine(AppContext.BaseDirectory, "images");

            if (!Directory.Exists(dir))

            {

                Directory.CreateDirectory(dir);

            }

            var filePath = Path.Combine(dir, $"{imgName}");

            await using (var fs = new FileStream(filePath, FileMode.Create, FileAccess.Write))

            {

                fs.Write(data, index, data.Length - index);

            }

            memory.Position = index;

            memory.Close();

            networkStream.Close();

        }

        private static int BinaryMatch(byte[] input, byte[] pattern)

        {

            var sLen = input.Length - pattern.Length + 1;

            for (var i = 0; i < sLen; ++i)

            {

                var match = true;

                for (var j = 0; j < pattern.Length; ++j)

                {

                    if (input[i + j] != pattern[j])

                    {

                        match = false;

                        break;

                    }

                }

                if (match)

                {

                    return i;

                }

            }

            return -1;

        }

        private static string GetHeaders(string path = "/")

            => $@"GET {path} HTTP/1.1

Host: {Host}

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/\*,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9

Connection: close

Pragma: no-cache

Upgrade-Insecure-Requests: 1

User-Agent: Programarea\_In\_Retea

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US,en-GB;q=0.9,en;q=0.8,ro-MD;q=0.7,ro;q=0.6,fr;q=0.5";

    }

}

**ÎNTREBĂRI LA APĂRAREA LABORATORULUI**

1. **Ce este un Socket ?**

Interfata socket este doar una dintre API-urile (application programming interfaces) catre protocoalele de comunicatie. Proiectat pentru a fi o interfata generica de programare a comunicatiei a fost prima dat introdus de catre sistemul UNIX  BSD 4.2. Cu toate ca nu a fost standardizat, a devenit un standard industrial de facto.

BSD 4.2 permitea doua domenii de comunicare diferite: Internet si UNIX. BSD 4.3 a adaugat protocoalele XNS (Xerox Network System) protocols, iar BSD 4.4 va adauga o interfata extinsa pentru a suporta protocoalele ISO OSI. Sa consideram urmatoarele terminologii:

* Un socket este un tip special de descriptor de fisier (file handle), care este folosit de un proces pentru a solicita servitii de retea de la sistemul de operare.
* O adresa socket address este un triplet:

**{protocol, adresa locala, proces local}**

In suita TCP/IP, spre exemplu:

**{tcp, 193.44.234.3, 12345}**

* O conversatie este o legatura de comunicatie dintre doua procese.
* O asociatie este un cvintet care specifica complet doua procese cuprinse intr-o conexiune:

**{protocol, adresa locala, proces local, adresa straina, proces strain}**

In suita TCP/IP, urmatoarea poate fi o asociatie valida:

**{tcp, 193.44.234.3, 1500, 193.44.234.5, 21}**

* O semi-asociatie este atat:

**{protocol, adresa locala, proces local}**

cat si

**{protocol, adresa straina, proces strain}**

care specifica fiecare jumatate a unei conexiuni.

* Semi-asociatia este deasemenea numita socket sau adresa de transport. Adica, un socket este un punct final pentru comunicatie care poate fi numit si adresat intr-o retea.

Doua procese comunica via socketuri TCP. Modelul socket asigura o conexiune full-duplex prin care doua procese isi pot trimite unul celuilalt mesaje sub forma unor fluxuri de octeti (byte streams). Aplicatia nu trebuie sa se preocupe de controlul celor doua fluxuri; aceste facilitati sunt asigurate de catre TCP. Ca si UDP, TCP foloseste porturi bine-cunoscute sau efemerale. Fiecare parte a unei conexiuni TCP poate fi identificata de tripletul

**<TCP, adresa IP, numar port>.**

Daca doua procese comunica prin TCP, ele au o conexiune logica care este unic identificabila de cele doua socketuri implicate, adica de catre combinatia

**<TCP, adresa IP locala, port local, adresa IP departata, port indepartat>**

1. **Scopul protocolului http**

Hypertext Transfer Protocol (HTTP) este metoda cea mai des utilizată pentru accesarea informațiilor în Internet care sunt păstrate pe servere World Wide Web (WWW). Protocolul HTTP este un protocol de tip text, fiind protocolul "implicit" al WWW. Adică, dacă un URL nu conține partea de protocol, aceasta se consideră ca fiind http. HTTP presupune că pe calculatorul destinație rulează un program care înțelegele protocolul. Fișierul trimis la destinație poate fi un document HTML (abreviație de la HyperText Markup Language), un fișier grafic, de sunet, animație sau video, de asemenea un program executabil pe server-ul respectiv sau și un editor de text. După clasificarea după modelul de referință OSI, protocolul HTTP este un protocol de nivel aplicație. Realizarea și evoluția sa este coordonată de către World Wide Web Consortium (W3C).

1. **Ce este un fir de execuție ?**

Conceptul de thread (fir de execuție) definește cea mai mică unitate de procesare ce poate fi programată spre execuție de către sistemul de operare. Este folosit în programare pentru a eficientiza execuția programelor, executând porțiuni distincte de cod în paralel în interiorul aceluiași proces. Câteodata însă, aceste portiuni de cod care constituie corpul threadurilor, nu sunt complet independente și în anumite momente ale execuției, se poate întampla ca un thread să trebuiască să aștepte execuția unor instructiuni din alt thread, pentru a putea continua execuția propriilor instrucțiuni. Această tehnică, prin care un thread asteaptă execuția altor threaduri înainte de a continua propria execuție, se numește sincronizarea threadurilor.

1. **Diferența dintre TCP si UDP**

**TCP** este cel mai frecvent utilizat protocol pe Internet.

Când solicitați o pagină web în browserul dvs., computerul trimite pachete TCP pe adresa serverului web, solicitându-i să trimită pagina Web înapoi la dvs. Serverul web răspunde prin trimiterea unui flux de pachete TCP, pe care browserul dvs. web le creează împreună pentru a forma pagina web. Când faceți clic pe un link, vă conectați, postați un comentariu sau faceți altceva, browserul dvs. web trimite pachete TCP către server și serverul trimite pachetele TCP înapoi.

TCP este vorba despre fiabilitatea-pachetele trimise cu TCP sunt urmărite astfel încât nu se pierde sau se pierde date în tranzit. Din acest motiv, descărcările de fișiere nu devin corupte, chiar dacă există sughițe în rețea. Desigur, dacă destinatarul este complet offline, calculatorul dvs. va renunța și veți vedea un mesaj de eroare care spune că nu poate comunica cu gazda de la distanță.

TCP realizează acest lucru în două moduri. În primul rând, ordonează pachetele numerotându-le. În al doilea rând, se verifică erorile prin faptul că destinatarul trimite un răspuns înapoi către expeditor, spunând că a primit mesajul. Dacă expeditorul nu primește un răspuns corect, acesta poate retrimite pachetele pentru a se asigura că destinatarul le primește corect.

Procesorul Explorer și alte utilitare de sistem pot afișa tipul de conexiuni pe care le produce un proces - aici vedem browserul Chrome cu conexiuni TCP deschise către o varietate de servere web.

Protocolul **UDP** funcționează similar cu TCP, dar aruncă toate chestiile de verificare a erorilor. Întreaga comunicare înapoi și înapoi introduce latență, încetinind lucrurile în jos.

Când o aplicație utilizează UDP, pachetele sunt trimise doar destinatarului. Expeditorul nu așteaptă să se asigure că destinatarul a primit pachetul - pur și simplu trimite în continuare pachetele următoare. Dacă destinatarul pierde câteva pachete UDP aici și acolo, acestea sunt doar pierdute - expeditorul nu le va retrimite. Pierderea tuturor acestor cheltuieli înseamnă că dispozitivele pot comunica mai repede.

UDP este utilizat atunci când viteza este de dorit și corectarea erorilor nu este necesară. De exemplu, UDP este frecvent utilizat pentru emisiuni live și jocuri online.

De exemplu, să presupunem că vizionați un flux video live, care sunt difuzate adesea folosind UDP în loc de TCP. Serverul trimite doar un flux constant de pachete UDP la calculatoarele care urmăresc. Dacă vă pierdeți conexiunea timp de câteva secunde, videoclipul poate îngheța sau se poate sărați pentru un moment, apoi treceți la bitul curent al emisiunii. Dacă întâmpinați pierderi de pachete minore, video sau audio pot fi distorsionate pentru o clipă, deoarece videoclipul continuă să fie redat fără datele lipsă.

Acest lucru funcționează similar în jocurile online. Dacă pierdeți unele pachete UDP, este posibil ca caracterele player-ului să teleporteze pe hartă, pe măsură ce primiți pachetele UDP mai noi. Nu are rost să cereți vechile pachete dacă le-ați pierdut, deoarece jocul continuă fără tine. Tot ce contează este ceea ce se întâmplă acum pe serverul de joc - nu ceea ce sa întâmplat cu câteva secunde în urmă. Scăderea corecției de eroare a TCP ajută la accelerarea conexiunii la joc și reducerea latenței.

1. **Ordinea în care octeții sunt secvențializați în rețea**

Un octet este o secvență de biți . În rețelele de calculatoare, unele protocoale de rețea trimit și primesc date sub formă de secvențe de octeți. Acestea sunt numite protocoale orientate de octeți . Exemple de protocoale orientate de byte includ TCP / IP și telnet .

Ordinea în care octeții sunt secvențializați într-un protocol de rețea octet orientat se numește comandă de byte de rețea . Dimensiunea maximă a unei singure unități de transmisie pentru aceste protocoale, unitatea maximă de transmisie (MTU) , este, de asemenea, măsurată în octeți. Programatorii de rețea lucrează în mod obișnuit atât cu comenzi de octeți de rețea, cât și cu MTU-uri.

Bytes-urile sunt utilizate nu numai în rețele, ci și pentru discuri de calculator, memorie și unități centrale de procesare (CPU-uri). În toate protocoalele de rețea moderne, un octet conține opt biți. Câteva computere (în general depășite) pot utiliza octeți de dimensiuni diferite pentru alte scopuri.

Secvența de octeți din alte părți ale computerului nu poate urma ordinea byte-ului din rețea. O parte a sarcinii subsistemului de rețea al unui computer este de a converti între ordinea octetului gazdă și comanda byte-ului de rețea atunci când este necesar.

1. **Cele patru nivele ale stivei TCP/IP și șapte nivele ale modelului OSI**

Modelul ISO-OSI cuprinde urmatoarele niveluri:

APLICATIE

PREZENTARE

SESIUNE

TRANSPORT

RETEA

LEGATURA DE DATE

FIZIC

Stiva TCP/IP cuprinde urmatoarele niveluri:

APLICATIE

TRANSPORT

RETEA

ACCES LA MEDIU

1. **Din ce cauză nu puteți transmite cereri HTTP website-ului utm.md utilizînd portul 80**

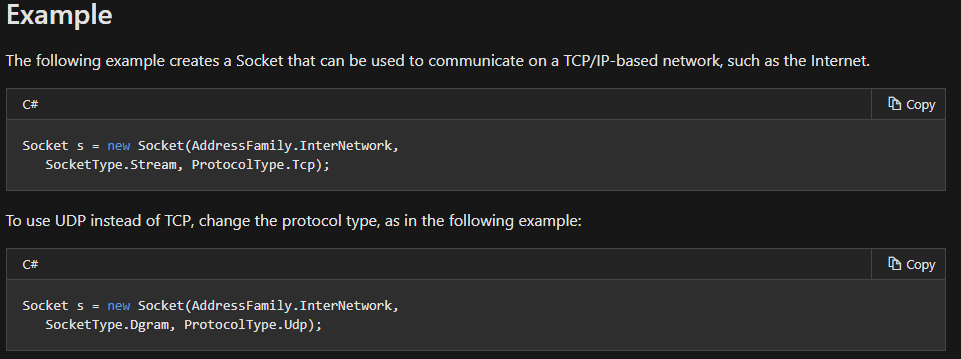
Nu putem transmite cereri http website-ului utm.md prin portul 80 fiindca utm.md foloseste conexiune prin certificate SSL si utilizeaza portul 443 pentru comunicare

1. **Cum se creează o conexiune Socket ?**

Socket-ii se utilizeaza intr-un sistem client/server. O aplicatie de tip server creeaza un socket si ii asociaza un nume: pentru socket-ii UNIX acesta va fi o intrare in sistemul de fisiere, iar pentru socket-ii de retea va fi un identificator de forma adresa IP + port. In continuare server-ul asteapta conexiuni de la clienti.

In momentul in care se accepta conexiunea, se creeaza un nou socket diferit de cel pe care s-au asteptat conexiuni. Acesta este folosit exclusiv pentru comunicatia cu clientul, primul socket putand astepta, in continuare, conexiuni de la clienti. In acest fel se pot folosi mai multe conexiuni.

Partea de client este mult mai directa. Se creeaza un socket si apoi se realizeaza conexiunea cu server-ul folosind pe post de adresa numele socket-ului server.



<https://docs.microsoft.com/en-us/dotnet/framework/network-programming/how-to-create-a-socket>

1. **Cum se transmit date intr-un Socket ?**

Socket-urile se pot folosi într-unul din două moduri posibile:

socluri neconectate;

socluri conectate (orientate pe conexiune).

În cazul soclurilor neconectate, se transmit pachete în protocolul UDP (User Datagram Protocol) între procese între care nu există o legătură permanentă, iar pachetele de date (datagramele - datagrams) sunt independente între ele și conțin adresa destinației pentru a putea fi rutate.

În cazul socket-urilor conectate, se stabilește mai întâi o legătură între două socket-uri aparținînd unor procese distincte (posibil plasate în stații diferite, conectate printr-o rețea de comunicație). Transmisia pe socluri conectate folosește protocolul TCP (Transmission Control Protocol) care este un protocol fiabil (cu verificări ale datelor transmise şi recepționate), spre deosebire de protocolul UDP care este un protocol nefiabil.

Pentru ca mai multe aplicații să poată avea acces la rețea în același timp, pachetele pentru fiecare aplicație trebuie să fie separate. Acest lucru se face cu ajutorul porturilor de comunicații. Un port este reprezentat de un număr pe 16 biți și fiecare aplicație folosește un port unic pentru a comunica în rețea. O conexiune are două porturi: cel local, alocat de sistemul de operare la inițierea conexiunii, și cel al server-ului, care este constant, în funcție de aplicație. Fiecare protocol standard are asociat un port conform unei liste menținute de cei de la Internet Assigned Numbers Authority. Ca exemple:

port 20 - File Transfer Protocol (date)

port 21 - File Transfer Protocol (control)

port 21 - Secure Shell (SSH)

port 25 - Simple Mail Transfer Protocol (SMTP)

port 80 - Hyper Text Transfer Protocol (HTTP)

etc.

1. **Cum se citesc datele dintr-un Socket ?**

<https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.receive?view=netframework-4.8>

The Receive() method of the socket class can be used to receive data. Here's how you can use it to retrieve data from a socket. Note that both the Send and Receive methods are blocking, i.e., they would block the currently executing thread till data has been sent or received.

byte[] data = new byte[1024];

int i = socketClient.Receive (data);

Note that you should incorporate the System.Net and System.Net.Sockets namespaces in your program to work with sockets.

using System.Net;

using System.Net.Sockets;

1. **Care sunt metodele HTTP?**

Metodele disponibile sunt :

GET: este cea mai folosită metodă, fiind utilizată atunci când serverului i se cere o resursă.

HEAD: se comportă exact ca metoda GET, dar serverul returnează doar antetul resursei, ceea ce permite clientului să inspecteze antetul resursei, fără a fi nevoit să obțină și corpul resursei.

PUT: metoda este folosită pentru a depune documente pe server, fiind inversul metodei GET.

POST: a fost proiectată pentru a trimite date de intrare către server.

DELETE: este opusul metodei PUT.

TRACE: este o metodă folosită de obicei pentru diagnosticare, putând da mai multe informații despre traseul urmat de legătura HTTP, fiecare server proxy adăugându-și semnătura în antetul Via.

OPTIONS: este folosită pentru identificarea capacităților serverului Web, înainte de a face o cerere.

CONNECT: este o metodă folosită în general de serverele intermediare.

1. **Codurile de stare HTTP (200, 301, 302, 401, 404, 405, 500)**

Codurile de stare al raspunsurilor HTTP sunt clasificate în 5 clase (categorii). Acestea sunt (pentru fiecare clasa sunt date câteva dintre erorile conținute):

1xx - erori informaționale: această clasă a status-ului indică un răspuns provizoriu al serverului și conține numai linia de status (de răspuns) sau alte aplicații opționale. Nu sunt aplicații necesare pentru acestă clasa de răspuns/status. Aceste status-uri pot fi ignorate.

100 - continuă:

Utilizatorul ar trebui să își continue cererea/acțiunea. Acest răspuns provizoriu este folosit pentru a informa utilizatorul că partea inițială a cererii a fost receptată și că deocamdată nu a fost refuzată de server. Utilizatorul ar trebui să continue și să ignore acest răspuns.

101 - schimbare protocol:

Server-ul înțelege și are intenția de a de a îndeplini cererea utilizatorului, răspunând prin această eroare că părți ale server-ului sunt în proces de schimbare/mutare. Server-ul va schimba protocolul imediat după ce răspunsul pentru linia 101 va fi terminată. Protocolul ar trebui schimbat doar în momentul în care acest lucru este avantajos și se permite.

2xx - răspuns reușit: clasa de răspuns/status ce indică utilizatorului că cererea a fost primită, înțeleasă și acceptată cu succes.

200 - ok:

Această cerere a fost executată cu succes. Informația a revenit cu un răspuns pozitiv, indiferent de modul în care s-a făcut cererea.

201 - creat/realizat:

Cererea a fost îndeplinită având ca rezultat crearea unui nou rezultat. Noul rezultat poate fi referit/raportat de către URI-uile înapoiate la intrarea răspunsului.

202 - acceptat:

Cererea a fost acceptata pentru procesare, dar aceasta din urmă nu a fost terminată complet. Scopul acestui mesaj este de a permite unui server să accepte cereri de la alți utilizatori, fără a cere conexiunii clientului să insiste până ce procesul/cererea e completă.

203 - informație neautorizată:

Informația returnată/revenită nu e definitivă ca fiind din server-ul principal, ci e adunata/receptionata de la un server local.

204 - fara continut:

Server-ul a indeplinit cererea si nu e nevoit sa intoarca raspunsul, dar ar dori sa raspunda printr-o informație recentă, gen meta.

205 - conținut refăcut:

Cererea a fost îndeplinita și ar trebui ca browser-ul să poată modifica/reseta modul de vizualizare a documentului ce a cauzat această cerere către server.

206 - conținut parțial:

Serverul a îndeplinit parțial cererea de primire de la sursă.

3xx - redirectări: această clasă de răspuns/status indică faptul că acțiunile următoare vor trebui făcute de browser pentru a putea fi îndeplinita cererea. Cererea ar putea fi direcționată de browser fără a interacționa cu utilizatorul dacă și numai dacă metoda folosită în cea de a doua cerere este „Primit/recepționat” sau „Direcționat/condus”.

300 - diferite/multiple alegeri:

Sursa cererii corespunde unor seturi de descrieri, fiecare cu locații specifice, iar browser-ul - dat fiind negocierea informașiei, primețte răspunsul astfel încât utilizatorul/browser-ul să poată alege modul dorit astfel încât redirectarea să fie spre acea locație. În cazul în care cererea a fost de tip „Condus/trimis”, răspunsul ar trebui să includă o intrare cu lista caracteristicilor și locațiilor de unde utilizatorul sau browser-ul poate alege sursa cea mai apropiată.

301 - modificat/mutat permanent:

Cererii i-a fost atribuite o sursă nouă și permanentă URI iar cererile următoare ar trebui să folosească una din sursele returnate URI. Dacă acest mesaj/cod este primit ca răspuns al unei cereri tip „Primit/recepționat” sau „Direcționat/condus”, browser-ul nu trebuie să redirecționeze automat cererea, doar dacă nu poate fi confirmată de către utilizator.

302 - găsit:

Sursa cererii este temporar pe un alt URI. În cazul în care redirectarea ar putea fi schimbată ocazional, utilizatorul ar trebui să folosească în continuare cererea URI (Request-URI) în cazul unor cereri ulterioare.

Dacă mesajul/statusul 302 este recepționat ca răspuns al unei cereri alta decât „Primit/recepționat” sau „Direcționat/condus”, browser-ul nu trebuie să redirecteze automat cererea dacă aceasta nu poate fi confirmată de cărte utilizator.

303 - vezi alta sursă:

Răspunsul cererii poate fi găsit sub un diferit URI și ar trebui să fie recepționat folosind metoda „Primit/recepționat” de la acea sursă.

304 - nemodificat:

În cazul în care clientul a efectuat o cerere de tip „Primit/recepționat” și accesul este permis, dar documentul nu a fost modificat, serverul ar trebui să răspundă cu acest mesaj/status.

305 - folosește proxy:

Cererea trebuie accesată printr-un proxy dat de către câmpul de locație. Acesta este dat de către URI. Beneficiarul va repeta acestă unică cerere prin intermediul unui proxy. Răspunsul 305 va fi generat doar de către serverul de origine.

306 (nefolosit):

Acest mesaj/status a fost folosit într-o vesiune anterioară a specificațiilor deci nu mai este folosit, el fiind reținut.

307 - redirectare temporară:

Sursa cerută se află temporar la un diferit URI. Din moment ce redirectarea poate fi modificata ocazional, utilizatoarul ar trebui să continuie să folosească URI-ul cerut pentru viitoarele acțiuni.

4xx - erori ale utilizatorilor: această clasă de mesaje/statusuri este folosită în cazurile în care utilizatorul ar putea greși formularea cererii. Excepția fiind răspunsurule pentru cererile tip „Direcționat/condus”, atunci serverul ar trebui să conțină o intrare cu o explicație a situației erorii și dacă e o eroare temporară sau pemanentă. Aceste răspunsuri sunt aplicabile pentru orice fel de cerere. Browser-ele ar trebui să arate orice intrare cerută de utilizator.

400 - cerere greșită:

Cererea nu a putut fi înțeleasă/percepută de către server din cauza unei sintaxe greșite/incomplete. Utilizatorul ar trebui să nu repete cererea fără ca aceasta să suporte modificări.

401 - neautorizat:

Cererea necesită autentificarea/înregistrarea utilizatorului. Răspunsul trebuie să includă WWW - câmp de autentificare conținând o somație aplicabilă utilizatorului. Utilizatorul poate repeta cererea. Dacă cererea deja includea câmpul de autorizare, atunci raspunsul 401 indică faptul că autorizarea a fost refuzată. Dacă răspunsul 401 conține aceeași somație ca răspuns principal iar browser-ul a executat autentificarea cel puțin o dată, atunci utilizatorul ar trebui să i se prezinte intrarea dată în răspuns, din moment ce aceasta include informații relevante.

402 - necesită plata:

Rezervat pentru utilizare ulterioară.

403 - interzis:

Serverul a înțeles cererea, dar refuză să o îndeplinească. Autorizarea nu ajută în nici un caz, iar cererea nu ar mai trebui repetata.

404 - negăsit:

Serverul nu a găsit nimic care să corespundă cererii URI. Nu se dau indicații despre condiția temporară sau permanentă.

405 - metodă nepermisă:

Metoda specificată în linia de cerere (Request-Line) nu este permisă de către sursa identificată după URI-ul cerut.

406 - neacceptat:

Sursa identificată de cerere este capabilă să genereze doar intrări care au conținut specific neacceptat de către condițiile de acceptare trimise prin cerere.

407 - autentificare prin proxy:

Mesajul este similar celui 401, dar indică situația în care utilizatorul trebuie să se autentifice prin proxy.

408 - cerere expirată:

Utilizatorul nu a făcut cererea în timpul în care serverul era pregătit sa o aștepte. Se poate repeta cererea fără modificări prealabile.

5xx - erori de server: răspunsurile/status-urile ce încep cu unitatea/cifra 5 indică cazurile în care serverul e conștient de greșelile produse sau e incapabil să execute cererea. Excepție facând răspunsul unei cereri „Direcționat/condus”, serverul ar trebui, în acest caz sa includă o afișare cu o explicație a situației de eroare, fie că e temporara sau permanentă.

500 - eroare internă de server:

Server-ul a întâmpinat o condiție neașteptată și o previne spre a putea îndeplini cererea.

501 - neîndeplinit/nerecunoscut:

Server-ul nu poate suporta funcționalitatea cerută pentru a putea îndeplini cererea. Acesta este răspunsul specific în cazurile în care server-ul nu recunoaște metoda de cerere și nu e capabil să o suporte indiferent de mijloc/resursă.

502 - poartă/ieșire greșită:

Server-ul, în timp ce încerca să se comporte ca o poartă/ieșire sau ca un proxy, a recepționat un răspuns invalid de la un server de deasupra/anterior în încercarea de a îndeplini cererea.

503 - serviciu nedisponibil:

În prezent server-ul nu poate să se ocupe de cererile trimise din cauza unei supraîncărcări sau a serviciilor de întreținere a server-ului ce au loc în acel moment. Aceasta este o stare temporară și în curând va fi remediată.

504 - poartă/ieșire expirată:

Server-ul, în timp ce încerca să se comporte ca o poartă/ieșire sau ca un proxy, nu a primit un răspuns la timp de la serverele de deasupra/anterioare lui specificat de URI (ex. HTTP, FTP, LDAP) sau de la un server auxiliar (ex. DNS) necesar accesării în încercarea de a termina/îndeplini cererea.

505 - versiunea HTTP nu e suportată/susținută:

Serveru-l nu suportă sau refuză să suporte versiunea de protocol a HTTP ce a fost folosită în formularea cererii. Server-ul sugerează că e incapabil să completeze/termine cererea folosind aceeași versiune cu cea a clientului.

1. **Ce este HTTP securizat ?**

HTTPS este o extensie a protocolului HTTP, care asigură o conexiune criptată. Acesta generează, la fiecare sesiune de comunicare, o cheie criptografică unică. Cheia respectivă poate fi decriptată doar de browserul utilizatorului și serverul pe care sunt stocate informațiile solicitate de utilizator. Astfel, loginurile și parolele utilizatorului, mesajele lui personale, datele cardurilor bancare și orice alte informații introduse de utilizator devin inaccesibile pentru terți.

Prelucrarea datelor are loc automat, iar utilizatorul nu trebuie să facă nimic în plus. Browserele moderne suportă tehnologia HTTPS, de aceea nu apar probleme cu procesarea cererilor. Protecția suplimentară este asigurată cu ajutorul certificatului SSL.

1. **Diferența dintre HTTP și HTTPS**

HTTP si HTTPS sunt doua protocoale diferite prin care putem transmite informatii catre server. In acest articol am sa va invat diferenta dintre HTTP si HTTPS. Cand scriem o adresa web in browser folosim automat unul dintre aceste protocoale, fie HTTP fie HTTPS. Putem sa aflam ce protocol folosim uitandu-ne la adresa , daca contine in fata ‘http://’ sau ‘https://’.

Hypertext Transfer Protocol (HTTP) : In mod normal este folosit protocolul HTTP pentru a trimite cereri catre server. in acest caz adresa web va incepe cu ‘http://’ si va folosi portul 80 pentru a comunica cu serverul. Acest protocol nu foloseste nici-un fel de criptare a datelor, deci nu este recomandat sa folosim acest protocol pentru trimiterea informatiilor importante (date personale, date card, date cont bancar, etc).

Hypertext Transfer Protocol Secure (HTTPS) :HTTPS este de fapt o combinatie intredoua p[rotocoale diferite, Hypertext Transfer Protocol (HTTPS) si SSL/TLS. LInk-urile incep cu ‘https://’. Acest protocol ne ofera o metoda sigura de a transmite datele importante.Acest protocol foloseste portul 443 pentru a comunica cu serverul. HTTPS are un algoritm de criptare ce asigura siguranta datelor.

1. **Diferența dintre Socket și WebSocket**

Chiar dacă realizează (în general) lucruri similare, da , sunt cu adevărat diferite. WebSockets rulează de obicei din browserele care se conectează la Application Server printr-un protocol similar cu HTTP care trece TCP / IP . Deci sunt în principal pentru aplicații web care necesită o conexiune permanentă la serverul său. Pe de altă parte, prize simple sunt mai puternice și generice. Se execută peste TCP / IP, dar nu sunt restricționate la browsere sau protocolul HTTP . Ele ar putea fi folosite pentru a implementa orice fel de comunicare.

1. **Diferența dintre un proces și un fir de execuție**

**Un proces** este o instanță a unui program de calculator care este executat. Acesta conține codul programului și activitatea curentă. În funcție de sistemul de operare (OS), un proces poate fi alcătuit din mai multe fire de execuție care execută instrucțiuni simultan. Procesul de multitasking vă permite să executați compilatorul Java în același timp în care folosiți un editor de text. În folosirea mai multor procese cu un singur procesor, se folosește contextul care comută între diferite contexte ale memoriei. Fiecare proces are un set complet de variabile proprii.

**Un thread** este o unitate de bază a utilizării procesorului, constând dintr-un contor de programe, o stivă și un set de registre. Un fir de execuție rezultă dintr-o furculiță a unui program de calculator în două sau mai multe sarcini care execută concomitent. Implementarea thread-urilor și proceselor diferă de la un sistem de operare la altul, dar în cele mai multe cazuri, un thread este conținut în interiorul unui proces. În cadrul aceluiași proces pot exista mai multe fire și pot partaja resurse, cum ar fi memoria, în timp ce diferite procese nu împărtășesc aceste resurse. Un exemplu de fire în același proces este verificarea ortografică automată și salvarea automată a unui fișier în timpul scrierii. Firele sunt, practic, procese care se execută în același context de memorie. Firele pot partaja aceleași date în timpul executării

1. **Ce este un Deadlock și când el apare ?**

În calculul simultan , un Deadlock este o stare în care fiecare membru al grupului așteaptă un alt membru, inclusiv el însuși, să ia măsuri, cum ar fi trimiterea unui mesaj sau eliberarea mai obișnuită a unui blocaj . Deadlock este o problemă comună în sistemele de multiprocesare , calcul paralel și sisteme distribuite , în care blocările de software și hardware sunt folosite pentru a arbitra resursele partajate și pentru a implementa sincronizarea proceselor .

Într-un sistem de operare , un blocaj are loc atunci când un proces sau thread intră într-o stare de așteptare , deoarece o resursă de sistem solicitată este deținută de un alt proces de așteptare, care la rândul său așteaptă o altă resursă deținută de un alt proces de așteptare. Dacă un proces nu poate să-și schimbe starea la nesfârșit, deoarece resursele solicitate de acesta sunt utilizate de un alt proces de așteptare, se spune că sistemul se află în impas.

Într-un sistem de comunicații , blocajele apar mai ales din cauza semnalelor pierdute sau corupte, mai degrabă decât a conținutului resurselor.

1. **Pentru ce este nevoie de metoda join() în programarea multithreading?**

Metoda join () care permite unui thread să aștepte până când un alt thread își termină execuția. Dacă t este un obiect Thread al cărui fir se execută în prezent, atunci t.join () se va asigura că t este încheiată înainte ca următoarea instrucțiune să fie executată de program.

Dacă există mai multe fire care apelează metodele join (), înseamnă că supraîncărcarea la unire permite programatorului să specifice o perioadă de așteptare. Totuși, la fel ca în cazul somnului, alăturarea depinde de sistemul de operare pentru cronometrare, deci nu ar trebui să presupuneți că unirea va aștepta exact atâta timp cât specificați.