**Програмування клієнт-серверних додатків**

Ціль роботи

– навчитись будувати розподілені системи на основі клієнт-серверної моделі.

Теоретичні відомості

### Мережеві сокети та порти

Мережевий сокет (network socket) – це кінцева точка двох сторонньої комунікаційної взаємодії комп’ютерів в мережі.

Інтерфейсний сокет (socket API) – програмний інтерфейс, який надає операційна система, та дозволяє програмним додаткам керувати та використовувати мережеві сокети.

Адреса сокету (socket address) – поєднання IP-адреси та номера порту. Базуючись на цій адресі інтернет-сокети доставляють відповідному програмному процесу чи потоку пакети даних, що надходять з мережі.

Слід розрізняти клієнтські і серверні сокети. Клієнтські сокети можна порівняти з кінцевими апаратами телефонної мережі, а серверні – з комутаторами. Клієнтський додаток, наприклад браузер, використовує тільки клієнтські сокети, а серверний, наприклад веб- сервер, якому браузер посилає запити – як клієнтські, так і серверні сокети.

Інтерфейс сокетів вперше з’явився в BSD Unix. Програмний інтерфейс сокетів описаний в стандарті POSIX.1 і в тій чи іншій мірі підтримується всіма сучасними операційними системами. Кожен процес може створити прослуховуючий сокет або серверний сокет і прив’язати його до якого-небудь порту операційної системи (рис.8.1). Процес, що прослуховує канал, зазвичай знаходиться в циклі очікування, тобто прокидається при появі нового з’єднання. При цьому зберігається можливість перевірити наявність з’єднань на даний момент, встановити тайм-аут для операції.

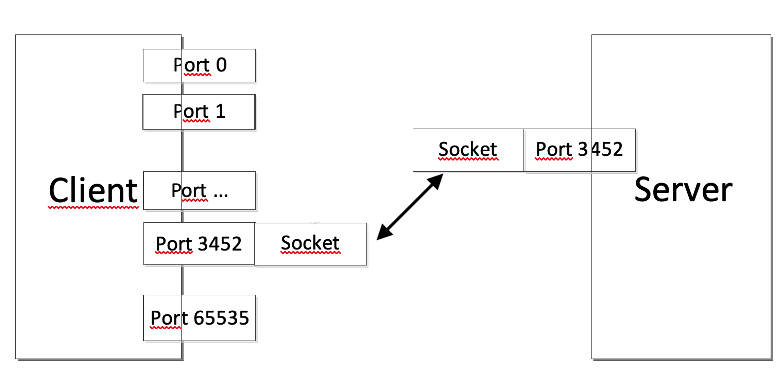


Рис.8.1. Загальна схема роботи сокетів

Кожен сокет має свою адресу. Операційні системи сімейства UNIX можуть підтримувати багато типів адрес, але обов’язковими є INET- адреса і UNIX-адреса. Якщо прив’язати сокет за UNIX-адресою, то буде створений спеціальний файл або файл сокету по заданому шляху, через який зможуть «спілкуватися» будь-які локальні процеси шляхом читання/запису з нього. Сокети типу INET доступні з мережі і вимагають виділення номера порту. Зазвичай клієнт явно під’єднується до слухача, після чого будь-яке читання або запис через його файловий дескриптор буде передаватися між ним і сервером. Загальновідомі сокети представляють собою зручний механізм апріорного прив’язування адреси сокету до якого завгодно стандартного сервісу. Наприклад, процес-сервер для програми Telnet жорстко зв’язаний з конкретним сокетом (рис.8.2). Адреса сокету може бути зарезервована для доступу до процедури перегляду, яка могла б вказувати сокет, крізь який можна було б отримати новоутворені послуги.

В протоколах TCP/UDP порт – це системний ресурс, який має номер та виділяється програмі для зв’язку з додатками, що виконуються на інших мережевих хостах (табл. 8.1). Порт може бути зайнятий тільки однією програмою і в цей час не може використовуватися іншою. Всі програми для зв’язку між собою за допомогою мережі використовують порти (до 65536 портів).

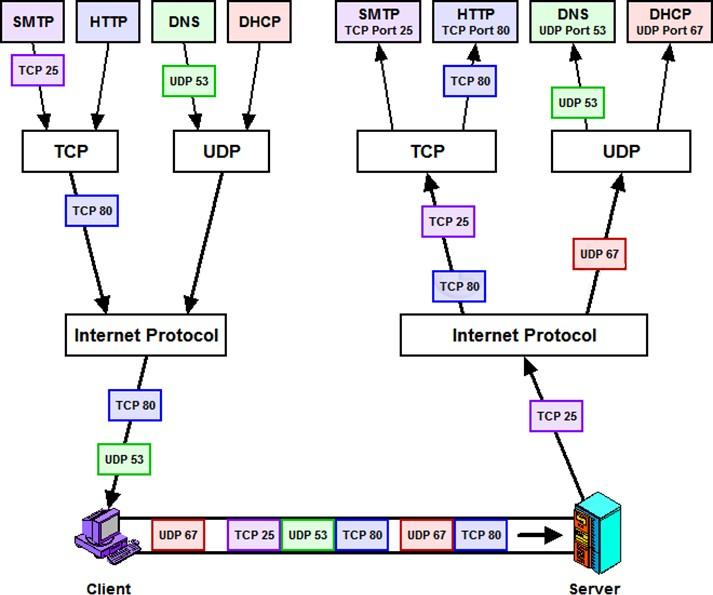


Рис.8.2 – Пояснення схеми роботи портів

Таблиця 8.1 – Деякі загальновідомі порти

| Порт | TCP | UDP | Опис |
| --- | --- | --- | --- |
| 0 | √ | √ | Зарезервовано |
| 20 | √ |  | Протокол FTP (дані) |
| 21 | √ |  | Протокол FTP (команди) |
| 22 | √ | √ | Протокол SSH |
| 23 | √ | √ | Протокол Telnet |
| 53 |  | √ | Протокол DNS |
| 80 | √ | √ | Протокол HTTP |
| 443 | √ |  | Протокол HTTPS |
| 6969 | √ |  | Клієнт BitTorrent |
| 33434 | √ | √ | Службова утиліта Traceroute |

### Стек протоколів TCP/IP. Протоколи TCP та UDP

TCP/IP – це абревіатура терміну Transmission Control Protocol / Internet Protocol (Протокол керування передачею / міжмережевий протокол). Фактично TCP/IP не один протокол, а декілька (табл. 8.2). Саме тому ви часто чуєте, як його називають набором, або комплектом протоколів, серед яких TCP і IP – два основні. Фактично TCP/IP представляє цей базовий набір протоколів, відповідальний за розбивання вихідного повідомлення на пакети (TCP), доставку пакетів на вузол адресата(IP) і збирання (відновлення) вихідного повідомлення з пакетів (TCP).

Таблиця 8.2 – Співвідношення моделі OSI та стеку протоколів TCP/IP

| OSI Model | TCP/IP Model | |
| --- | --- | --- |
| Application | Application | Telnet, SMTP, POP3, FTP, HTTP, SNMP, DNS, SSH, … |
| Presentation |
| Session |
| Transport | Transport | TCP, UDP |
| Network | Internet | IP, ICMP, ARP, DHCP |
| Data Link | Network Access | Ethernet, ADSL, … |
| Physical |

### Прикладний рівень

Протоколи прикладного рівня TCP/IP визначають процедури стосовно організації взаємодії прикладних процесів (програм) різних мережевих комп’ютерів і форми подання інформації за такої взаємодії. По ознакам взаємодії прикладних процесів виділяють два типи прикладного програмного забезпечення: програма-клієнт та програма- сервер. Протоколи прикладного рівня зорієнтовано на конкретні прикладні завдання. Серед традиційних послуг, котрі забезпечують протоколи прикладного рівня із сімейства TCP/IP, сьогодні найпопулярнішими є електронна пошта – протоколи SMTP та POP3, передавання файлів – FTP та TFTP, емуляції віддаленого терміналу – TELNET тощо.

В інтернеті активно використовуються послуги, які базуються на технології WWW, яка ґрунтується на протоколі передавання гіпертексту HTTP. Сьогодні є популярні послуги пакетної IP-телефонії на базі стандартів IETF, яку стосуються спеціальних протоколів прикладного, транспортного і мережевого рівнів, наприклад сигналізації SIP, передавання в режимі реального часу RTP та RTCP, резервування ресурсів RSVP, рекомендацій ITU H.323 тощо.

### Транспортний рівень

Протоколи транспортного рівня TCP/IP надають транспортні послуги прикладним процесам. Основними протоколами транспортного рівня TCP/IP є протокол керування передаванням TCP (Transmission Control Protocol) і протокол користувальницьких дейтаграм UDP (User Datagram Protocol). Транспортні послуги цих протоколів суттєво відрізняються. Протокол UDP доставляє дейтаграми без установлення з’єднання. При цьому він не гарантує їхнього доставляння. Протокол TCP забезпечує надійне доставляння байтових потоків (сегментів) із попереднім встановленням транспортного дуплексного з’єднання (віртуального каналу) між модулями TCP мережевих комп’ютерів. Для розв’язання транспортних завдань протоколи TCP та UDP в перебігу передавання даних формують і додають до даних свої заголовки обсягом 20 байт та 8 байт відповідно.

Кожен прикладний процес взаємодіє з модулем транспортного рівня TCP або UDP через окремий порт, що дозволяє при взаємодії систем однозначно ідентифікувати прикладні процеси. Ці порти нумеруються починаючи з нуля. При передаванні запиту прикладної програми клієнта до прикладної програми сервера транспортний модуль, формуючи дейтаграму чи сегмент, вказує номери портів програмних модулів прикладних протоколів сервера й клієнта. З цією метою в заголовку пакета протоколу транспортного рівня виділено два поля – «порт одержувача» і «порт відправника», обсягом по 2 байти. Номери портів TCP та UDP до прикладних протоколів сервера стандартизовано IETF. Для цього надано номери в діапазоні від 1 до 1023. Наприклад, програмний модуль TСP сервера взаємодіє з модулем протоколу HTTP через порт з номером 80. Взаємодія модуля TCP чи UDP клієнта з будь-яким модулем прикладного протоколу відбувається через порт, якому надається вільний номер, за значенням більший ніж 1023.

### Мережевий рівень

Протоколи мережевого рівня TCP/IP забезпечують взаємодію поміж мережами різної архітектури тощо. Основним протоколом мережевого рівня технології TCP/IP є міжмережевий протокол IP та його допоміжні протоколи: адресний протокол ARP; реверсний адресний протокол RARP (Reverse ARP); протокол діагностичних повідомлень ICMP (Internet Control Message Protocol), який надсилає повідомлення вузлам мережі про помилки на маршруті, які виникають при передаванні пакетів тощо.

Головне завдання міжмережевого протоколу IP – це маршрутизація пакетів даних поміж різнотипними комп’ютерними мережами. Для розв’язання цього завдання протокол IP підтримує IP-адресацію мереж та вузлів, використовує таблицю маршрутизації пакетів, виконує, за необхідності, фрагментацію та дефрагментацію цих пакетів.

Функціонування мережевого рівня також забезпечує низка протоколів динамічної маршрутизації RIP, OSPF, які динамічно формують маршрути таблиці маршрутизації за алгоритмами вектора VDA (Vector Distance Algorithm) і стану зв’язку LSA (Link State Algorithm) відповідно, протоколів політики зовнішньої маршрутизації EGP (Exterior Gateway Protocol), BGP (Border Gateway Protocol) тощо.

Засоби мережевого рівня забезпечують доставку даних між пристроями в складових мережі, а саме комп’ютерами, маршрутизаторами і т.д. Однак не слід забувати, що на одному вузлі може функціонувати паралельно декілька програм, яким потрібен доступ до мережі. Отже, дані всередині комп’ютерної системи повинні розподілятися між програмами. Тому, при передачі даних по мережі недостатньо просто адресувати конкретний вузол. Необхідно також ідентифікувати програму-одержувача, що неможливо здійснити засобами мережевого рівня.

Іншою серйозною проблемою протоколів мережевого рівня є відсутність засобів, що дозволяють передавати великі масиви даних. Коли вихідні дані перевищують максимально допустимий розмір пакета мережевого рівня, то ці дані повинні бути розбиті на порції, кожна з яких передається в мережу окремим пакетом. Проте кожен пакет мережевого рівня передається по мережі як єдиний, незалежний від інших блок даних. У разі якщо будь-які пакети "загубилися", то модуль мережевого протоколу на приймаючій стороні не зможе виявити втрату, і, отже – виявити порушення цілісності загального масиву даних. Тому кошти транспортного рівня забезпечують відсутність втрат інформації. Такий режим передачі даних отримав назву гарантованої доставки.

Таким чином, засоби транспортного рівня представляють собою функціональну надбудову над мережевим рівнем і вирішують дві основні задачі:

1. забезпечення доставки даних між конкретними програмами, що функціонують, в загальному випадку, на різних вузлах мережі;
2. забезпечення гарантованої доставки масивів даних довільного розміру.

В даний час в Інтернет використовуються два транспортних протоколу – UDP, що забезпечує негарантовану доставку даних між програмами, і TCP, що забезпечує гарантовану доставку з встановленням віртуального з’єднання.

Для ідентифікації програм протоколи транспортного рівня в мережі Інтернет (TCP і UDP), використовують унікальні числові значення, так звані порти. Номери портів призначаються програмами відповідно до її функціонального призначення на основі певних стандартів. Для кожного протоколу існують стандартні списки відповідності номерів портів і програм. Так, наприклад, програмне забезпечення WWW, що працює через транспортний протокол TCP, використовує TCP-порт 80, модулі протоколу FTP – TCP-порт 21, а служба DNS взаємодіє з транспортними протоколами TCP і UDP через TCP-порт 53 і UDP-порт 53 відповідно.

Таким чином, протокол мережевого рівня IP і транспортні протоколи TCP і UDP реалізують дворівневу схему адресації: номери TCP-і UDP-портів дозволяють однозначно ідентифікувати програму в рамках вузла, а сам вузол однозначно визначається IP-адресою. Отже, комбінація IP-адреси і номера порту дозволяє однозначно ідентифікувати програму в мережі Інтернет. Така комбінована адреса називається сокетом або socket.

Протокол UDP (User Datagram Protocol) – протокол транспортного рівня, що входить в стек протоколів TCP/IP, що забезпечує негарантовану доставку даних без встановлення віртуального з’єднання (рис. 8.3).

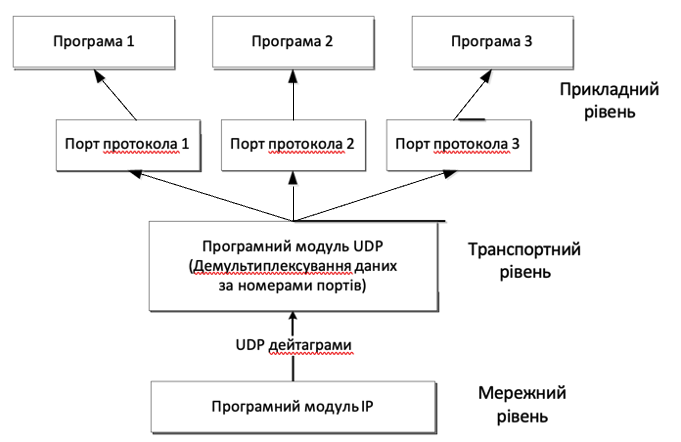


Рис.8.3 Робота додатку за UDP протоколом

Оскільки на протокол не покладається завдань щодо забезпечення гарантованої доставки, а лише потрібно забезпечувати зв’язок між різними програмами, то структура заголовка дейтаграми UDP, таку назву має пакет протоколу, виглядає досить просто – вона включає в себе всього чотири поля. Перші два поля містять номери UDP-портів програми-відправника та програми-одержувача. Два інших поля в структурі заголовка дейтаграми призначені для управління обробкою – це загальна довжина дейтаграми і контрольна сума заголовка.

Протокол TCP (Transmission Control Protocol) є транспортним протоколом стека протоколів TCP/IP, що забезпечує гарантовану доставку даних з встановленням віртуального з’єднання (рисунок 4.4). Протокол надає програмам, що використовують його, можливість передачі безперервного потоку даних. Дані, що підлягають відправці в мережу,розбиваються на порції, кожна з яких забезпечується службовою інформацією, тобто формуються пакети даних. У термінології TCP пакет називається сегментом.

Відповідно до функціонального призначення протоколу структура TCP-сегмента передбачає наявність наступних інформаційних полів:

1. номер порту-відправника і номер порту-одержувача – номери портів, що ідентифікують програми, між якими здійснюється взаємодія;
2. поля, призначені для забезпечення гарантованої доставки: розмір вікна, номер послідовності і номер підтвердження;
3. керуючі прапори – спеціальні бітові поля, що управляють протоколом.

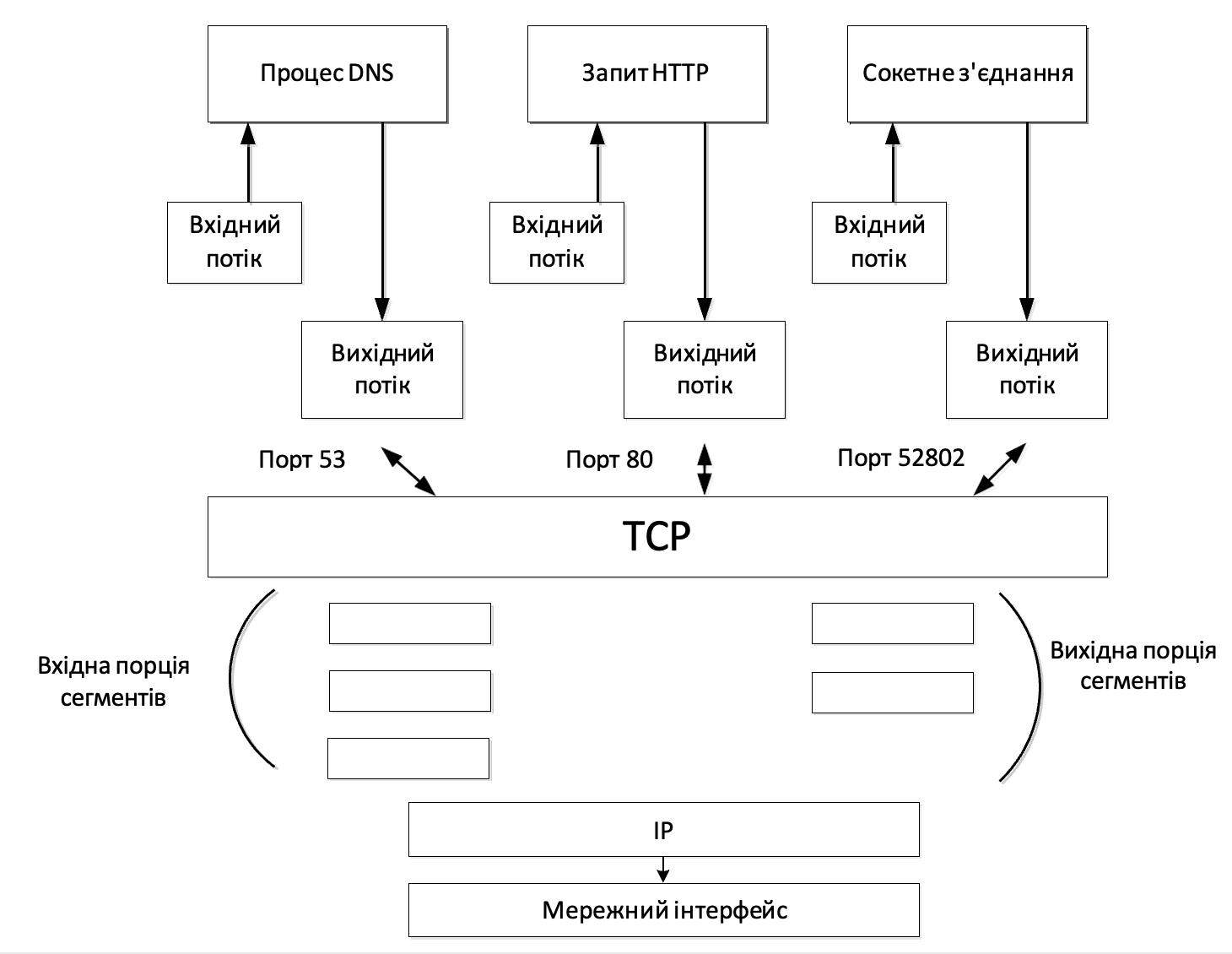


Рис.8.4 Робота додатку за TCP протоколом

Для забезпечення гарантованої доставки протокол TCP використовує механізм відправки підтвердження. З метою зниження завантаження мережі протокол TCP допускає посилку одного підтвердження відразу для декількох отриманих сегментів. Обсяг даних, які можуть бути передані в мережу відправником до отримання підтвердження, визначається спеціальним параметром протоколу TCP – розміром вікна. Розмір вікна узгоджується при встановленні з’єднання між відправником та одержувачем і може автоматично змінюватися програмними модулями протоколу TCP в залежності від стану каналу зв’язку. Якщо в процесі передачі даних втрати відбуваються досить часто, то розмір вікна зменшується, і навпаки – вікно може мати великий розмір, якщо висока надійність каналу даних. Для того, щоб дані могли бути правильно зібрані одержувачем в потрібному порядку, в заголовку TCP-сегмента присутня інформація, яка визначає положення вкладених даних в загальному потоці. Відправляючи підтвердження, одержувач вказує положення даних, які він очікує отримати в наступному сегменті, тим самим побічно повідомляючи відправнику, який фрагмент загального потоку був успішно прийнятий. Відповідні поля заголовка TCP-сегмента отримали назву номер послідовності і номер підтвердження.

Таблиця 8.3 – Порівняння протоколів TCP та UDP

| TCP | UDP |
| --- | --- |
| Надійний протокол | Ненадійний протокол |
| Встановлення віртуального  з’єднання | Без встановлення віртуального  з’єднання |
| Повторна передача сегментів та  керування потоком | Без повторної передачі  повідомлень |
| Використовується  впорядкування сегментів | Випадковий порядок отримання  сегментів |
| Для контролю слугують  сегменти підтвердження | Підтвердження передачі відсутнє |

Перед початком передачі потоку даних абоненти повинні узгодити параметри передачі: розмір вікна та початкові номери послідовностей, щодо яких буде відраховуватися положення переданих в сегментах даних усередині загального потоку. Очевидно, що таке узгодження передбачає обмін спеціальними сегментами і виділення ресурсів, зокрема, блоків пам’яті, необхідних для прийому та обробки даних і підтверджень. Відповідна послідовність дій називається встановленням віртуального з’єднання.

### Мережеві засоби платформи .NET Framework

У таблиці 8.4 коротко подано перелік класів бібліотеки .NET, з використанням яких можна побудувати мережевий додаток.

Таблиця 8.4. – Основні класи для роботи з сокетами в .NET

| Клас .NET | Опис |
| --- | --- |
| Socket | Забезпечує базову функціональність сокета для  додатка |
| TcpClient | Побудований на класі Socket, щоб забезпечити ТСР обслуговування на більш високому рівні. TcpClient надає декілька методів для відправки та  отримання даних мережею |

Базовим класом при побудові мережевих додатків є клас System.Net.Sockets.Socket, деякі властивості та методи якого представлено в таблиці 8.5.

Таблиця 8.5. – Короткий опис класу System.Net.Sockets.Socket

| Властивості та  методи | Короткий опис та призначення |
| --- | --- |
| AddressFamily | Сімейство адрес сокета (значення із перерахунку  Socket.AddressFamily) |
| Available | Об’єм даних, які можна зчитати |
| Blocking | Чи знаходиться сокет в блокуючому режимі |
| Connected | Чи з’єднаний сокет з віддаленим хостом |
| LocalEndPoint | Локальна кінцева точка сокета |
| ProtocolType | Тип протокола сокета |
| RemoteEndPoint | Віддалена кінцева точка сокета |
| SocketType | Тип сокета |
| Accept() | Створює новий сокет для обробки вхідного запиту  на зєднання |
| Bind() | Звязує сокет з локальною кінцевою точкою для  очікування вхідних запитів на зєднання |
| Close() | Закриває сокет |
| Connect() | Встановлює з’єднання з віддаленим хостом |
| Listen() | Переводить сокет в режим прослуховування |
| Receive() | Отримує дані від приєднаного сокета |

Робоче завдання

1. Реалізувати індивідуальне завдання. Засіб розробки можна обирати на свій розсуд
2. Задокументувати у звіті по виконаній роботі висновки за результатами досліджень і підготувати відповіді на контрольні питання.

Методичні вказівки

TCP сервер, побудований з використанням класу Socket

static void Main(string[] args) { //Sockets.NET Server

IPEndPoint ipEndPoint =

new IPEndPoint(IPAddress.Parse("127.0.0.1"), 8000);

Socket serverSock = new Socket(AddressFamily.InterNetwork,

SocketType.Stream, ProtocolType.Tcp);

serverSock.Bind(ipEndPoint);

Console.WriteLine("Очікування вхідного TCP підключення...");

serverSock.Listen(10);

Socket clientSock = serverSock.Accept();

IPEndPoint remote = (IPEndPoint)clientSock.RemoteEndPoint;

Console.WriteLine("Зєднання встановлено за {0}.

Вхідне повідомленн:", remote.Address.ToString());

byte[] bufRecieve = new byte[8];

clientSock.Receive(bufRecieve);

Console.WriteLine(Encoding.UTF8.GetString(bufRecieve));

Console.WriteLine("Press any key for finish...");

Console.ReadKey();

serverSock.Close();

clientSock.Close();

}

TCP клієнт, побудований з використанням класу Socket

static void Main(string[] args) { //NET.Socket Client

Socket cliSock = new Socket(AddressFamily.InterNetwork,

SocketType.Stream, ProtocolType.Tcp);

IPAddress ipServ = IPAddress.Parse("127.0.0.1");

IPEndPoint ipEndP = new IPEndPoint(ipServ, 8000);

try {

cliSock.Connect(ipEndP);

} catch (SocketException ex) {

Console.WriteLine(ex);

}

byte[] pack = Encoding.UTF8.GetBytes("Hello");

cliSock.Send(pack);

Console.WriteLine("Передавання завершено. Press any key for continue...");

Console.ReadKey();

cliSock.Close();

}

TCP сервер, побудований з використанням класу TcpListener

static void Main(string[] args) {//TCPListener .NET

TcpListener tcpServer = new TcpListener(IPAddress.Parse("127.0.0.1"), 8000);

tcpServer.Start();

Console.WriteLine(" Очікування підключення...");

TcpClient tcpClient = tcpServer.AcceptTcpClient();

Console.WriteLine("Підключення встановлено, вхідне повідомлення:");

byte[] buffer = new byte[8];

NetworkStream streamTcp = tcpClient.GetStream();

streamTcp.Read(buffer, 0, buffer.Length);

Console.WriteLine(Encoding.UTF8.GetString(buffer));

streamTcp.Close();

tcpClient.Close();

tcpServer.Stop();

Console.WriteLine("Press any key for finish... ");

Console.ReadKey();

}

TCP клієнт, побудований з використанням класу TcpListener

static void Main(string[] args) { //TCPClient .NET

TcpClient clientTcp = new TcpClient();

clientTcp.Connect(IPAddress.Parse("127.0.0.1"), 8000);

NetworkStream streamTcp = clientTcp.GetStream();

byte[] buffer = new byte[8];

buffer = Encoding.UTF8.GetBytes("Hello");

streamTcp.Write(buffer, 0, buffer.Length);

streamTcp.Close();

clientTcp.Close();

Console.WriteLine("Передавання завершено. Press any key for continue...");

Console.ReadKey();

}

Індивідуальні завдання.

Розробити клієнтський та серверний додаток на базі TCP протоколу. Засіб розробки можна обирати на свій розсуд.

| № варіанту | Завдання |
| --- | --- |
| 1 | Відправити з клієнта на сервер своє ім’я, на сервері додати до імені своє прізвище та відправити результат на клієнт. |
| 2 | Відправити на сервер два числа, знайти їх суму т повернути результат клієнту. |
| 3 | Відправити на клієнт два числа, знайти їх добуток та повернути результат серверу. |
| 4 | Відправити на сервер масив, знайти його максимальний елемент та повернути результат клієнту. |
| 5 | Відправити на сервер число, сервер перевіряє, чи є воно більшим за введене на ньому значення, якщо так, то повертає на клієнт своє число, інакше повертає клієнту його ж число. |
| 6 | Відправити на сервер число, сервер перевіряє, чи є це число паліндромом. Якщо так, то відправляє на клієнт «Yes», якщо ні, то повертає «No». |
| 7 | Відправити з сервера на клієнт масив чисел, відсортувати їх за зростанням та повернути на сервер. |
| 8 | Відправити на сервер повідомлення, знайти у ньому кількість літер «a», та відправити на клієнт результат. |
| 9 | Відправити на клієнт повідомлення, знайти його довжину і відправити відповідну кількість нулів на сервер. |
| 10 | Відправити повідомлення на сервер, перевірити чи є в ньому цифри, якщо так, то відправити повідомлення «Yes» на клієнт, інакше «No» |
| 11 | Відправити з клієнта на сервер своє прізвище, на сервері додати до прізвища своє ім’я та відправити результат на клієнт. |
| 12 | Відправити на сервер три числа, знайти їх суму та повернути результат клієнту. |
| 13 | Відправити на клієнт три числа, знайти добуток від’ємних чисел та повернути результат серверу. |
| 14 | Відправити на сервер масив, знайти його мінімальний та максимальний елементи та повернути результати клієнту |
| 15 | Клієнт відправляє чотири числа на сервер, сервер додає до чисел їх середнє значення та повертає клієнту. |
| 16 | Відправити на сервер число, сервер перевіряє, чи є це число простим. Якщо так, то відправляє на клієнт «Yes», якщо ні, то повертає «No» |
| 17 | Відправити з сервера на клієнт масив цілих чисел, який вводить користувач, відсортувати їх за спаданням та повернути на сервер. |
| 18 | Відправити на сервер повідомлення, знайти у ньому кількість пробілів та відправити на клієнт результат. |
| 19 | Відправити на клієнт повідомлення, знайти його довжину і відправити відповідну кількість символів «Х» на сервер. |
| 20 | Відправити повідомлення на сервер, перевірити чи є в ньому пробіли, якщо так, то відправити повідомлення «Yes» на клієнт, інакше «No» |
| 21 | Відправити на сервер число, розкласти його на прості множники, результат повернути клієнту у вигляді масиву. |
| 22 | Відправити на сервер число, сервер перевіряє, чи є це число квадратом іншого цілого числа. Якщо так, то відправляє на клієнт «Yes», якщо ні, то повертає «No». |
| 23 | Відправити з сервера на клієнт масив чисел, замінити від’ємні значення нулем та повернути на сервер. |
| 24 | Відправити на сервер повідомлення, знайти у ньому кількість голосних літер та відправити на клієнт результат. |
| 25 | Відправити на клієнт повідомлення, знайти його довжину і відправити відповідну кількість випадкових чисел на сервер. |
| 26 | Відправити повідомлення на сервер, перевірити чи містить воно знаки пунктуації, якщо так, то відправити повідомлення «Yes» на клієнт, інакше «No» |
| 27 | Відправити з клієнта на сервер рядок тексту, на сервері вилучити всі прописні літери, результат повернути клієнту. |
| 28 | Відправити на сервер три числа, знайти їх середнє геометричне та повернути результат клієнту. |
| 29 | Відправити на клієнт два повідомлення, якщо вони однакові, то серверу надіслати рядок тексту «Failed». |
| 30 | Відправити на сервер масив, знайти добуток його елементів та повернути результати клієнту |

Контрольні питання

1. Що таке мережевиий сокет?
2. Які основні принципи мережної взаємодії додатків із застосуванням портів?
3. Коротко охарактеризуйте стек протоколів TCP/IP.
4. Перерахуйте номери портів таких протоколів та служб: HTTP, DNS, FTP, Telnet, ICQ, Skype.
5. Як співвідноситься модель відкритої взаємодії OSI та стек протоколів TCP/IP?
6. Чим відрізняються протоколи транспортного рівня TCP та UDP?
7. Коротко опишіть схему створення сокетного TCP з’єднання.
8. Які засоби платформи .NET Framework призначені для передачі даних між вузлами комп’ютерної мережі?
9. Запропонуйте підхід до створення багатоклієнтського сервера на основі протокола TCP.
10. Яким чином можна реалізувати луна-сервер та визначити час передачі даних між вузлами?