Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна

ІНФОРМАЦІЙНІ МЕРЕЖІ

Методичні рекомендації до виконання
індивідуальних (розрахунково-графічних) робіт для здобувачів вищої освіти
першого (бакалаврського) рівня факультету математики і інформатики

Електронний ресурс

Харків – 2025

УДК 004.738+004.77
І 74

Рецензенти:

М. О. Момот – кандидат технічних наук, доцент, доцент закладу вищої освіти кафедри комп’ютерних наук та інформаційних технологій Національного аерокосмічного університету ім. М. Є. Жуковського «Харківський авіаційний інститут»;

Д. Ю. Узлов – кандидат технічних наук, доцент, директор Навчально-наукового інституту комп'ютерних наук та штучного інтелекту Харківського національного університету імені В. Н. Каразіна.

Затверджено до розміщення в мережі Інтернет рішенням Науково-методичної ради Харківського національного університету імені В. Н. Каразіна (протокол № 1 від 23 жовтня 2025 року)

І 74 **Інформаційні** мережі : методичні рекомендації до виконання індивідуальних (розрахунково-графічних) робіт для здобувачів вищої освіти першого (бакалаврського) рівня [Електронне видання] / К. М. Руккас, Н. І. Леонова, В. В. Фролов, К. В. Носов. – Харків : ХНУ імені В. Н. Каразіна, 2025. – (PDF 45 с.)

Методичні рекомендації містять теоретичний матеріал, завдання для індивідуальних (розрахунково-графічних) робіт, зразки виконання розрахунково-графічних робіт, питання для самоконтролю, рекомендовану літературу.

Навчально-методичне видання призначається для здобувачів факультету математики і інформатики першого (бакалаврського) рівня вищої освіти, які вивчають освітню компоненту «Інформаційні мережі».

УДК 004.738+004.77

ЗМІСТ

Вступ

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ до виконання індивідуальних (розрахунково-графічних) робіт з освітньої компоненти «Інформаційні мережі» для здобувачів вищої освіти факультету математики і інформатики першого (бакалаврського) рівня вищої освіти.

На сьогоднішній день будь-які процеси супроводжуються обміном інформацією, що забезпечується завдяки функціонуванню інформаційних мереж. Розвиток комп'ютерних технологій, глобалізація та зростання обсягів даних зумовлюють постійне вдосконалення архітектури, протоколів і технологій побудови мереж. Інформаційні мережі є основою для функціонування всіх інформаційних систем і сервісів, забезпечуючи надійність, безпеку та швидкість передачі даних.

Опанування освітню компоненту «Інформаційні мережі» дозволить студентам на професійному рівні засвоїти принципи побудови, функціонування і адміністрування комп'ютерних мереж, а також навчитися застосовувати відповідні протоколи, моделі та інструменти для забезпечення якісної мережевої інфраструктури.

Перед виконанням індивідуальних завдань студенти зобов'язані опрацювати основні теоретичні положення та виконати відповідні практичні завдання.

Індивідуальна робота № 1
ОРГАНІЗАЦІЯ ЛОКАЛЬНОЇ ОБЧИСЛЮВАЛЬНОЇ МЕРЕЖІ. АНАЛІЗ РОБОТИ ПРОТОКОЛУ TCP.

Теоретичний матеріал

1. Принципи IP-адресації.

У поточній (IPv4) реалізації адрес IP, IP адреси складаються з 4 байтів що дають разом 32 біта доступної інформації. Для зручності читання (і з організаційних причин) IP адреси зазвичай записуються в «точково-розділової нотації». Наприклад:

192.168.1.24 – чотири числа, розділені (.) точками.

IP-адреса складається з двох логічних частин: одна визначає ідентифікатор мережі, до якої належить вузол, інша – безпосередньо ідентифікує сам хост (мережевий інтерфейс) у межах цієї мережі. Співвідношення кількості бітів, відведених під мережевий і хостовий ідентифікатори, залежить від класу IP-адреси, що визначає її структуру та діапазон адресації.

Класи IP-адрес

Класова адресація розділяє IP-адреси на п’ять основних класів (A, B, C, D, E), які відрізняються обсягом доступних бітів для ідентифікації мереж і хостів.

IP-адреси ***класу A*** характеризуються тим, що для позначення ідентифікатора мережі використовуються перші 8 біт (перший октет), а решта 24 біти призначені для адресації хостів у межах цієї мережі. Адреси класу A мають найстарший біт першого байта, встановлений у 0, що відповідає діапазону значень першого октету 0 – 127. Отже, загальна кількість можливих мереж класу A становить 128, а кожна з них може налічувати до 2^{24} (16 777 216) адрес хостів.

Слід зазначити, що IP-адреси 0.0.0.0 та 127.0.0.0 мають спеціальне призначення і не можуть використовуватися як ідентифікатори звичайних мереж. Отже, реальна кількість доступних мереж класу A становить 126.

IP-адреси ***класу B*** характеризуються тим, що перші два октети (16 біт) використовуються для позначення ідентифікатора мережі, а решта 16 біт призначена для адресації хостів у цій мережі. Крім того, адреси класу B мають два найстарших біти першого октету рівними 10, що обмежує діапазон перших байтів значеннями від 128 до 191 у десятковому представленні. Це означає, що для адресації мережі залишається невикористаними 14 біт, що забезпечує до 16384 унікальних адрес мереж класу B, кожна з яких може

включати до 2^{16} (65536) інтерфейсів, включно з адресами службового призначення.

Клас C IP мережевих адрес використовує 24 біти (три старших байта) для ідентифікації мережі, що залишилися 8 біт (останній байт) вказує адресу хоста. Адреса класу З завжди має старші три біти встановленими в 110. Таким чином, для номера мережі залишається 21 біт, що дає (2097152) доступних мереж класу С. Перший октет адреси мережі класу С може приймати значення від 192 до 223, і кожна з таких мереж може мати до 28 254 доступних інтерфейсів.

IP-адреси ***класу С*** використовують перші три октети (24 біти) для ідентифікації мережі, і лише останній октет (8 біт) відводяться для адресації хостів. Старші три біти першого октету мереж класу С мають значення 110. В результаті, залишається 21 біт для формування номеру мереж, що дозволяє створити до 2097152 мереж, кожна з яких підтримує до 254 доступних інтерфейсів (не враховується широкомовна адреса та адреса самої мережі).

IP-адреси класів D та E мають діапазони значень першого байту з 224 по 239 та з 240 по 254 відповідно, проте в даний час вони не використовуються.

Таблиця 1.1. Діапазони IP-адрес за класами

Клас	**Найменша адреса**	**Найбільша адреса**
A	1.0.0.0	126.0.0.0
B	128.0.0.0	191.255.0.0
C	192.0.0.0.	223.255.255.0
D	224.0.0.0	239.255.255.255
E	240.0.0.0	247.255.255.255

Для організації локальних мереж, які можуть бути підключені до глобальної мережі Internet за допомогою механізмів трансляції адрес, у кожному з основних класів IP-адрес зарезервовано спеціальні діапазони, призначені для внутрішнього використання. Ці адреси не маршрутизуються в глобальній мережі та можуть вільно використовуватися всередині корпоративних або домашніх мереж без необхідності отримання унікальних глобальних IP-адрес.

Одна мережа класу А	10.0.0.0
16 мереж класу В	172.16.0.0 – 172.31.0.0
256 мереж класу С	192.168.0.0 – 192.168.255.0

Мережеві маски.

Маска – це 32-бітне двійкове число, яке використовується в IP-мережах для відокремлення мережевої частини IP-адреси від хостової частини. У двійковому поданні одиниці містяться тільки у тих розрядах, які відповідають за ідентифікацію мережі. Нижче наведені значення мережевих масок для основних класів IP-мереж:

Клас A (8 мережевих біт) 255.0.0.0
Клас B (16 мережевих біт) 255.255.0.0
Клас C (24 мережевих біта) 255.255.255.0

Використання мережевих масок.

Організація підмережі (subnetting) – це метод логічного поділу однієї IP-мережі на кілька менших, взаємопов'язаних локальних підмереж (сабнетів).. Застосування цього прийому дає змогу оптимізувати структуру адресного простору, підвищити ефективність маршрутизації та локалізувати мережевий трафік, зменшуючи навантаження на канали зв'язку між різними групами вузлів. Це особливо важливо в корпоративних мережах, де необхідне логічне розділення відділів, сервісів або зон безпеки.

Методика розподілу на підмережі полягає у запозиченні доступних хостових бітів та внесенні відповідних змін у мережеву маску, що змушує інтерфейси локально інтерпретувати ці запозичені біти як частину мережевих бітів. Для того, щоб розділити IP-мережу на дві локальні підмережі, треба запозичити один хостовий біт. Це реалізується шляхом модифікації значення мережевої маски, а саме зміною значення відповідного (запозиченого) біту з 0 на 1.

Розглянемо значення мережевої маски для мереж класу C, що дозволяє організувати дві локальні підмережі:

11111111.11111111.11111111.10000000 або 255.255.255.128

На прикладі IP-адреси 192.168.1.0, що належить до класу, розглянемо значення мережевої маски залежно від кількості підмереж.

Кількість хостів	Кількість підмереж	Мережева маска
2	126	255.255.255.128 (11111111.11111111.11111111.10000000)
4	62	255.255.255.192 (11111111.11111111.11111111.11000000)

8	30	255.255.255.224 (11111111.11111111.11111111.11100000)
16	14	255.255.255.240 (11111111.11111111.11111111.11110000)
32	6	255.255.255.248 (11111111.11111111.11111111.11111000)
64	2	255.255.255.252 (11111111.11111111.11111111.11111100)

2. Мережеві утиліти.

ARP – відображення та керування таблицею відповідностей між IP-адресами та MAC-адресами в локальній мережі.

IPCONFIG – отримання інформації про конфігурацію мережевих інтерфейсів комп'ютера.

NETSTAT – відображення інформації про сесію TCP/IP.

PING – перевірка доступності вузла в мережі шляхом надсилання ICMP-запитів.

ROUTE – відображає або модифікує таблицю маршрутизації.

TRACERT – перегляд та керування таблицею маршрутизації операційної системи.

Arp (Address Resolution Protocol)

Відображення та зміна ARP таблиці відповідності IP-адрес та фізичних адрес.

Якщо при зверненні до ARP-таблиці не вдається знайти відповідність між IP-адресою та фізичною адресою, то у мережу надсилається широкомовний ARP-запит (broadcast), що містить запитувану IP-адресу.

Широкомовний запит – це мережеве повідомлення, що надсилається усім пристроям у межах однієї локальної мережі. Воно має спеціальну MAC-адресу призначення FF:FF:FF:FF:FF:FF, яка означає, що пакет має бути прийнятий кожним пристроєм у мережевому сегменті, незалежно від його IP-адреси.

Кроки виконання ARP-запиту:

1. Відправник (наприклад, комп'ютер A) хоче надіслати IP-пакет до певної IP-адреси (наприклад, 192.168.1.10), але не знає MAC-адресу цієї IP-адреси.

2. В системі перевіряється ARP-таблиця (локальний кеш відповідностей IP та MAC). Якщо запису немає – ініціюється широкомовний ARP-запит.
3. У мережу надсилається широкомовний ARP-запит (тобто Ethernet-фрейм з MAC-адресою одержувача FF:FF:FF:FF:FF:FF), який містить питання, що можна інтерпретувати як: «Хто має IP-адресу 192.168.1.10? Надішліть мені свою MAC-адресу!»
4. Усі пристрої в мережі отримують цей запит, але відповість лише той, чия IP-адреса збігається з шуканою.
5. Відповідь надсилається унікальним (unicast) повідомленням до ініціатора: «Я – 192.168.1.10, моя MAC-адреса – XX:XX:XX:XX:XX:XX»
6. Відправник зберігає цю відповідність у своїй ARP-таблиці і надсилає IP-пакет з уже відомою MAC-адресою одержувача.

Синтаксис команд:
ARP -s inet_addr eth_addr [if_addr]
ARP -d inet_addr [if_addr]
ARP -a [inet_addr] [-N if_addr]

Таблиця 1.2. Параметри команди ARP

-a	основний параметр, який виводить усю ARP-таблицю або, якщо вказано конкретну IP-адресу ([inet_addr]), – записи, пов'язані лише з нею. У форматі виводу відображається список IP-адрес, їх відповідні MAC-адреси та тип запису (динамічний або статичний).
-g	Аналог -a
inet_addr	Конкретна IP-адреса вузла, для якого необхідно переглянути ARP-відповідність.
-N if_addr	Вказує локальну IP-адресу мережевого інтерфейсу, таблицю ARP якого слід переглянути. Це дозволяє дослідити ARP-таблиці кількох інтерфейсів (наприклад, у випадку наявності декількох мережевих адаптерів або IP-адрес на одному комп'ютері).
-d	Призначена для видалення запису з ARP-таблиці, що відповідає заданій IP-адресі (inet_addr). Застосування цієї команди доцільне у випадках, коли ARP-таблиця містить застарілі або некоректні записи, що можуть впливати на маршрутизацію або спричиняти конфлікти адрес.
-s	Додавання статичного запису до ARP-таблиці, вручну задаючи відповідність між IP-адресою та MAC-адресою. Статичні ARP-записи корисні в умовах, де необхідна жорстка фіксація відповідностей (наприклад, у середовищах з підвищеними

	вимогами до безпеки або при налагодженні мереж).
eth_addr	Відповідна MAC-адреса (у форматі xx-xx-xx-xx-xx-xx).
if_addr	Необов'язковий параметр, який вказує IP-адресу, для якої слід змінити таблицю адрес. Якщо параметр не встановлено, використовується перший доступний інтерфейс.

Ipconfig

Отримання інформації про мережеву конфігурацію та керування параметрами мережевих інтерфейсів.

Використання: ipconfig [/? | /all | /release [адаптер] | /renew [адаптер]]

Таблиця 1.3. Параметри команди ipconfig

/?	Виведення вікна довідки для цієї утиліти.
/all	Виведення повної інформації про налаштування.
/release	Звільнення IP-адреси вказаного адаптера.
/renew	Оновлення IP-адреси вказаного адаптера.

За замовчуванням відображаються лише IP-адреса, маска підмережі та основний шлюз для кожного адаптера, пов'язаного з TCP/IP.

Якщо для ключів Release та Renew не встановлено ім'я адаптера, то дозволено звільнятися або оновлюватися адреси IP для всіх адаптерів, пов'язаних з TCP/IP.

Netstat

Дозволяє переглядати поточні мережеві з'єднання, статистику протоколів і таблиці маршрутизації. Є інструментом діагностики мережі, який використовується для аналізу активних з'єднань TCP/UDP, виявлення відкритих портів, перегляду статистики протоколів, моніторингу мережевої активності та таблиці маршрутів.

NETSTAT [-a] [-e] [-n] [-s] [-p ім’я] [-r] [інтервал]

Таблиця 1.4. Параметри команди netstat

-a	Виводить усі з'єднання та порти, що прослуховуються (listening ports) – як TCP, так і UDP.
-e	Показує мережеву статистику (кількість байтів, пакетів, помилок) –

	для Ethernet-інтерфейсів. Можна використовувати разом з -s.
-n	Виводить адреси та порти у числовому вигляді, без спроби без спроби зіставлення IP-адрес із доменними іменами та портів із іменами служб.
-p ім’я	Обмежує вивід інформації до конкретного протоколу, наприклад TCP, UDP, ICMP тощо.
-r	Виводить таблицю маршрутизації, аналогічно до route print.
-s	Показує статистику для кожного протоколу (TCP, UDP, ICMP, IP); дає змогу аналізувати рівень протоколів (оцінити кількість вхідних і вихідних пакетів, виявити кількість помилок, відмов, повторних передач). За замовчуванням відображає статистику по протоколу TCP. Використання -p із зазначенням відповідного протоколу дозволяє отримати статистику по вказаному протоколу.
інтервал	Оновлює статистику через задану кількість секунд. В разі вказання цього параметру, оновлення відбувається постійно. Припини оновлення можна за допомогою комбінації CTRL+C. Без вказання інтервалу вивід інформації буде відбуватись один раз.

Ping

Використовується для перевірки доступності вузла в мережі та оцінки якості з’єднання

Синтаксис:

ping [-t] [-a] [-n число] [-l розмір] [-f] [-i TTL] [-v TOS] [-r число] [-s число] [[-j список_Вузлів] | [-k список_Вузлів]] [-w інтервал] список_Розсилки

Таблиця 1.5. Параметри команди ping

-t	Безперервне надсилання запитів до цільового вузла до ручного переривання (Ctrl+C).
-a	Визначає та виводить доменне ім’я вузла за IP-адресою.
-n число	Задає кількість запитів, які потрібно надіслати (за замовчуванням – 4)
-l розмір	Визначає розмір буфера запиту в байтах (тобто об’єм даних, що пересилається).

-f	Встановлює прапорець «Don’t Fragment» у пакеті – використовується для перевірки можливості фрагментації в маршруті.
-i TTL	Задає значення TTL (Time-To-Live) – максимальна кількість маршрутизаторів, які може пройти пакет.
-v TOS	Задає тип сервісу (TOS – Type of Service), що рідко використовується в сучасних IP-мережах.
-r число	Вказує кількість мережевих пристроїв (до 9), які записуються в заголовок пакету при проходженні маршруту. Використовується для відслідковування фактичного маршруту.
-s число	Визначає кількість часових позначок (timestamp), які потрібно записати в пакет (до 4).
-j список_Вузлів	Встановлює *вільний маршрут* (loose source route) – IP-адреси вузлів, через які пакет повинен пройти, але не обов’язково в послідовному порядку.
-k список_Вузлів	Встановлює *жорсткий маршрут* (strict source route) – IP-адреси, через які пакет має пройти точно в заданій послідовності.
-w інтервал	Задає час очікування відповіді в мілісекундах (таймаут).
список_Розсилки	IP-адреса або доменне ім’я вузла, до якого надсилаються запити

Route

Обробка таблиць мережних маршрутів.

ROUTE [-f] [-p] [команда [вузол] [MASK маска] [шлюз] [METRIC метрика] [IF інтерфейс]]

Таблиця 1.5. Параметри команди route

-f	Очищення таблиць маршрутів від записів для всіх шлюзів. При заданні однієї з команд, таблиці очищаються до виконання команди.

-p	При використанні з командою ADD визначає збереження маршруту при перезавантаженні системи. За замовчуванням маршрути не зберігаються під час перезавантаження. У разі використання з командою PRINT задає список зареєстрованих постійних маршрутів. Не використовується з іншими командами.
Команда	Одна з наступних команд:
PRINT	Друк маршруту
ADD	Додавання маршруту
DELETE	Видалення маршруту
CHANGE	Зміна існуючого маршруту
Вузол	Вузол, що адресується (мережа призначення)
MASK	Якщо вводиться ключове слово MASK, наступний параметр інтерпретується як значення мережевої маски. Значення маски підмережі, яке зв'язується із записом для даного маршруту. Якщо цей параметр не заданий, за замовчуванням мається на увазі 255.255.255.255.
Шлюз	Шлюз. IP-адреса наступного маршрутизатора (шлюзу), через який мають пересилатися пакети до зазначеної мережі.
IF	Інтерфейс
Інтерфейс	Номер мережевого інтерфейсу, через який буде здійснюватися передача.
METRIC	Метрика (або ціна) для вузла, що адресується. Менше значення – пріоритетніше.

Пошук всіх символічних імен вузлів проводиться у файлі бази даних NETWORKS. Пошук символічних імен шлюзів проводиться у файлі бази імен вузлів HOSTS.

Для команд PRINT та DELETE можна вказати вузол та шлюз за допомогою підстановочних знаків ('*' або '?'), або можна взагалі опустити параметр "шлюз".

Якщо у параметрі використовуються знаки * або ?, він інтерпретується як шаблон, і друкуються лише відповідні цьому шаблону маршрути. Знак '*' замінює будь-який рядок, а знак '?' замінює будь-який одиночний знак.

Приклади: 157.*.1, 157.*, 127.*, *224*.

Примітки щодо діагностики:

Неправильне значення мережевої маски призводить до помилки, це відбувається, якщо (ВУЗОЛ & МАСКА) != ВУЗОЛ.

Приклад > route ADD 157.0.0.0 MASK 155.0.0.0 157.55.80.1 IF 1

Помилка додавання маршруту: 87

Приклади:

> route PRINT

> route ADD 157.0.0.0 MASK 255.0.0.0 157.55.80.1 METRIC 3 IF 2

	вузол		маска	шлюз	значення метрики	інтерфейс

Якщо IF не вказано, виконується пошук найкращого інтерфейсу для цього шлюзу.

> route PRINT
> route PRINT 157* *//Друк тільки тих, що відповідають шаблону 157**
> route DELETE 157.0.0.0
> route PRINT

Tracert

Мережева утиліта командного рядка, призначена для діагностики та відстеження шляху проходження пакетів від вашого комп'ютера до цільового вузла в мережі.

Синтаксис команди:

tracert [-d] [-h макс_Число] [-j список_Вузлів] [-w інтервал] [-R] [-4] [-6] [-S адреса_джерела] цільовий_хост

Таблиця 1.5. Параметри команди tracert

-d	Забороняє перетворення IP-адрес на імена хостів. Це прискорює виведення результатів.
-h макс_Число	Визначає максимальну кількість проміжних вузлів у маршруті. Стандартне значення – 30.

-j список_Вузлів	Вільний вибір маршруту за списком вузлів. Не всі маршрутизатори підтримують цю функцію.
-w інтервал	Інтервал очікування кожної відповіді у мілісекундах. Стандартне значення – 4000мс (4 секунди)
-R	Вказує на трасування зворотного маршруту (доступне лише в IPv6).
-4	Примусово використовує IPv4 протокол.
-6	Примусово використовує IPv6 протокол.
-S адреса_джерела	Вказує вихідну IPv6-адресу, яка буде використана для надсилання пакетів.
цільовий_хост	IP-адреса або доменне ім’я одержувача, для якого виконується трасування маршруту.

Net

Синтаксис цієї команди:
NET [ACCOUNTS | COMPUTER | CONFIG | CONTINUE | FILE | GROUP | HELP |
HELPMSG | LOCALGROUP | NAME | PAUSE | PRINT | SEND | SESSION | SHARE | START | STATISTICS | STOP | TIME | USE | USER | VIEW]

Можна використовувати такі імена команд:

NET ACCOUNTS	NET HELP	NET SHARE
NET COMPUTER	NET HELPMSG	NET START
NET CONFIGNET	LOCALGROUPNET	STATISTICS
NET CONFIG SERVER	NET NAME	NET STOP
NET CONFIG WORKSTATION	NET PAUSE	NET TIME
NET CONTINUE	NET PRINT	NET USE
NET FILE	NET SEND	NET USER
NET GROUP	NET SESSION	NET VIEW

SERVICES – Ця команда виводить список служб, які можна запустити.

SYNTAX – ця команда виводить пояснення синтаксичних правил, що використовуються під час опису команд у Довідці.

3. Протокол TCP

Transmission Control Protocol – це протокол транспортного рівня моделі OSI/моделі TCP/IP, який забезпечує надійну передачу даних між

пристроями в комп'ютерній мережі. Він гарантує, що дані будуть доставлені повністю, у правильному порядку та без помилок.

Для ідентифікації конкретного процесу або служби на хості використовується ***порт*** – логічний номер (від 0 до 65535), що використовується разом з IP-адресою. У контексті TCP-протоколу, порти відіграють ключову роль у забезпеченні адресації процесів (додатків) на хості. Саме порти дають змогу TCP-протоколу розрізняти множинні паралельні з'єднання, які можуть одночасно існувати на одному вузлі.

За допомогою портів, TCP підтримує мультиплексування – тобто можливість передавати дані до/від різних процесів через одне фізичне мережеве з'єднання.

Таблиця 1.6.Класифікація портів

Діапазон портів	**Призначення**
0 – 1023	Зарезервовані (well-known) порти – для системних служб (HTTP, HTTPS, FTP, SSH)
1024 – 49151	Зареєстровані порти – для користувацьких додатків
49152 – 65535	Динамічні/приватні порти – зазвичай призначаються клієнтськими програмами динамічно

TCP-з'єднання унікально ідентифікується комбінацією чотирьох параметрів: IP-адреса відправника, порт відправника, IP-адреса одержувача, порт одержувача.

Комбінація IP адреси та номери порту іноді називається ***сокетом (socket)***. Цей термін виник у вихідній специфікації TCP (RFC 793).

Протокол TCP передає дані потоково – тобто у вигляді безперервного потоку байтів, які розбиваються на ***сегменти*** для транспортування мережею. На відміну від протоколів, які працюють з фіксованими повідомленнями (наприклад, UDP), TCP не зберігає межі повідомлень, а лише забезпечує доставку байтів у правильному порядку, без втрат і дублювання.

Умовно кажучи, якщо застосунок хоче передати 5000 байтів даних, TCP може розбити їх на 5 сегментів по 1000 байтів, пронумерувати кожен сегмент, і відстежувати, чи кожен із них був отриманий. Якщо один із сегментів загубився – він буде повторно переданий.

Контроль порядку і цілісності даних у TCP забезпечується відповідними полями заголовка TCP-сегмента. TCP-сегмент складається з заголовка (мінімум 20 байт) та корисного навантаження (даних).

<table>
<tr><td colspan="8">Source Port</td><td colspan="8">Destination Port</td></tr>
<tr><td colspan="16">Sequence Number (Seq)</td></tr>
<tr><td colspan="16">Acknowledgment Number (Ack)</td></tr>
<tr><td rowspan="2" colspan="4">Data Offset</td><td rowspan="2" colspan="3">Reserved</td><td colspan="6">Flags</td><td rowspan="2" colspan="8">Window Size</td></tr>
<tr><td>U
R
G</td><td>A
C
K</td><td>P
S
H</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td></tr>
<tr><td colspan="8">Checksum</td><td colspan="8">Urgent Pointer</td></tr>
<tr><td colspan="16">Options</td></tr>
<tr><td colspan="16">Data</td></tr>
</table>

Рис. 1.1. Структура TCP-сегмента

Source Port	Значення номера порту відправника.
Destination Port	Значення номера порту одержувача.
Sequence Number «Номер послідовності»	Числове значення порядкового номера першого байта даних у цьому сегменті.
Acknowledgment Number «Номер підтвердження»	Очікуваний наступний байт від одержувача.
Data Offset	Довжина TCP-заголовка (в 32-бітових словах).
Reserved	Зарезервовано, має бути нульовим.
Flags	Управляючі біти: URG, ACK, PSH, RST, SYN, FIN.
Window Size	Розмір приймального вікна для керування потоком передачі даних.
Checksum	Контрольна сума для перевірки помилок.
Urgent Pointer	Вказує на пріоритетні дані (якщо URG встановлено)
Options	Додаткові параметри (напр., масштабування вікна)
Data	Безпосередньо інформація, що передається.

Однією з ключових властивостей TCP є ***забезпечення гарантованої доставки*** повідомлень ***у правильному порядку, без втрат або дублювання***. Цього вдається досягти за рахунок використання порядкових номерів байтів, механізмів підтвердження (ACK), а також спеціальних прапорів керування у заголовку TCP-сегмента.

Оскільки, кожен сегмент містить у заголовку порядковий номер (Sequence Number) – це дозволяє приймаючій стороні правильно впорядковувати дані, навіть якщо сегменти надходять непослідовно.

Підтвердження доставки реалізується через поле Acknowledgment Number, яке містить номер наступного байта, що очікується. Якщо,

наприклад, приймач отримав усі дані до байта 500 включно, він надішле підтвердження з номером 501. Таким чином, відправник отримує зворотній зв’язок щодо статусу доставки.

Початок TCP-сесії відбувається за стандартним триетапним ***протоколом встановлення з’єднання***, відомим як потрійне рукостискання (three-way handshake). У першій фазі клієнт надсилає сегмент з прапором SYN та початковим порядковим номером. У відповідь сервер надсилає сегмент з прапорами SYN та ACK – підтвердження та свій порядковий номер. Після чого клієнт відправляє сегмент з прапором ACK, і з’єднання вважається встановленим.

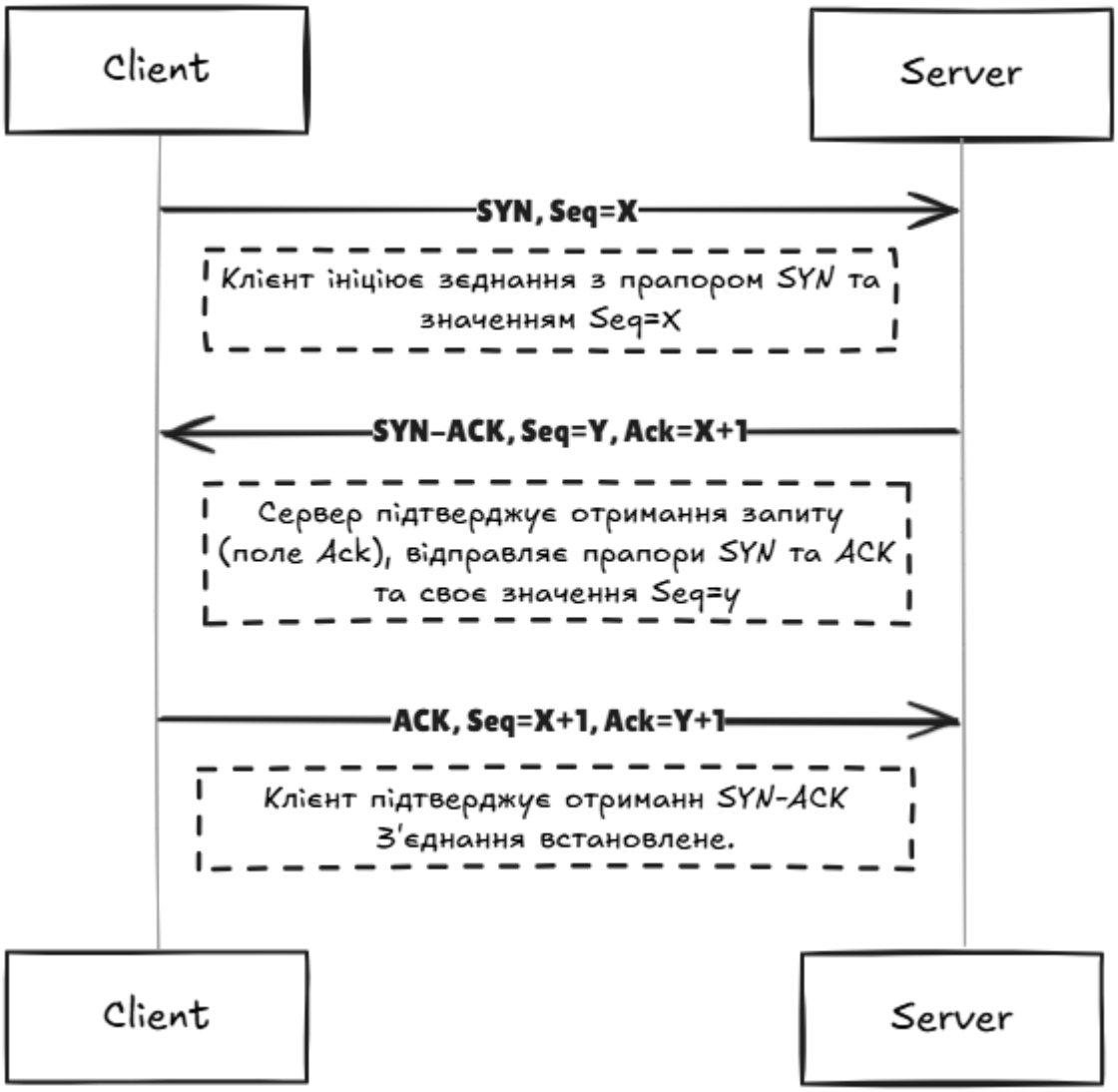

Рис. 1.2. Алгоритм встановлення TCP з’єднання

Після встановлення з’єднання починається ***передача даних.*** У разі втрати пакета через ненадходження підтвердження TCP ініціює повторну передачу. Передані дані не зникають до отримання відповідного ACK, що гарантує надійність і цілісність.

На рисунку 1.3 схематично наведено приклад етапу передачі даних під час TCP з’єднання. У цьому прикладі клієнт відправляє серверу сегмент з

порядковим номером 150 та розміром даних Len=K. Сервер, своєю чергою, не передає інформацію, а відправляє тільки, так звані, «квитанції підтвердження», тобто сегменти з встановленим прапором ACK та з пустим полем Data. В разі успішної доставки інформації від клієнта, значення поля Acknowledgment Number (Ack) буде сумою порядкового номера переданого сегмента та розміру переданих даних. Слід зазначити, що передача інформації можлива в обидві сторони. Тоді будуть змінюватись значення відповідних полів у сегментах не тільки клієнта, а і сервера.

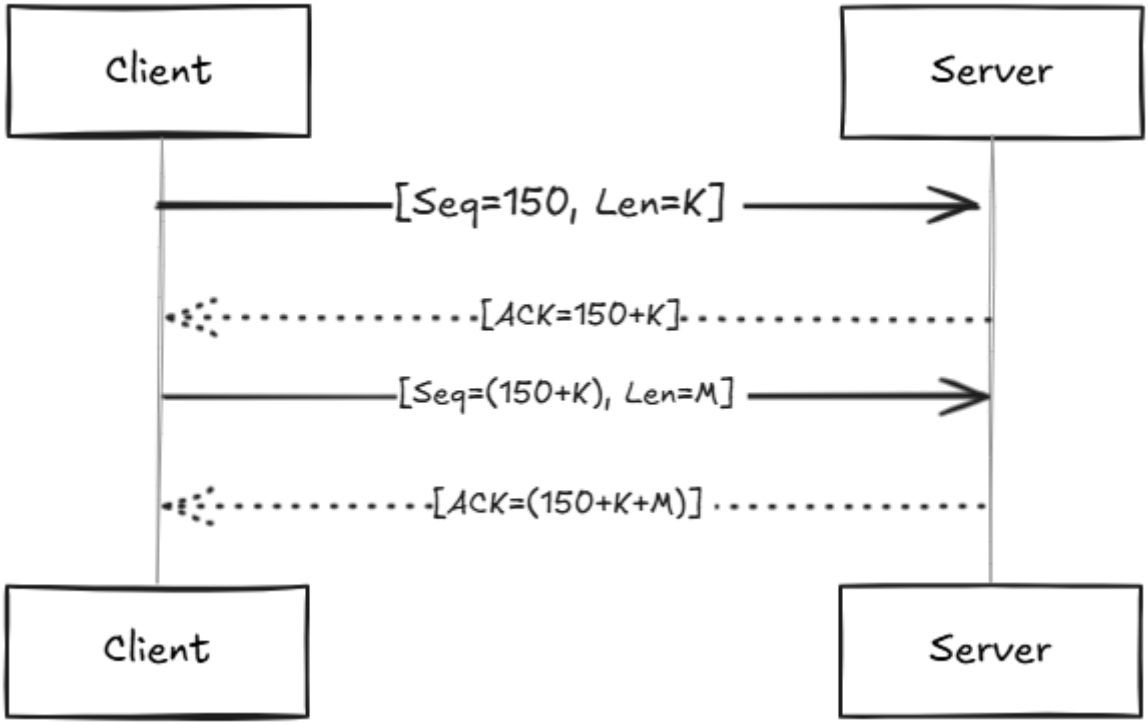

Рис. 1.3. Приклад алгоритму передачі даних

Окрему увагу слід звернути на поле Window, яке визначає кількість байтів, які приймач готовий прийняти без підтвердження. TCP використовує ***механізм ковзного вікна***, при якому відправник може надсилати кілька сегментів поспіль. Після отримання ACK вікно «зсувається» вперед, дозволяючи передавати нові дані. Це забезпечує ефективне використання мережевих ресурсів і контроль за перевантаженням приймача.

Під час передавання інформації важливу роль відіграють прапори (flags) заголовка TCP, зокрема:

- ACK – підтвердження отримання даних;
- SYN – ініціалізація з’єднання;
- FIN – завершення з’єднання;
- RST – аварійне скидання з’єднання;
- PSH – примусова доставка даних до прикладного рівня;

- URG – вказує на наявність термінових даних.

Встановлений прапор PSH (Push Function) означає, що приймач повинен негайно передати отримані дані з буфера TCP до прикладного рівня, минаючи звичайну буферизацію. Це важливо для програм, що працюють у реальному часі – наприклад, у термінальних сесіях або інтерактивних застосунках.

Завершення TCP-з'єднання також має чітко визначену процедуру: одна зі сторін надсилає FIN, інша – підтверджує ACK, а потім надсилає свій FIN і знову очікує ACK. Такий чотирикроковий механізм забезпечує правильне завершення обміну даними.

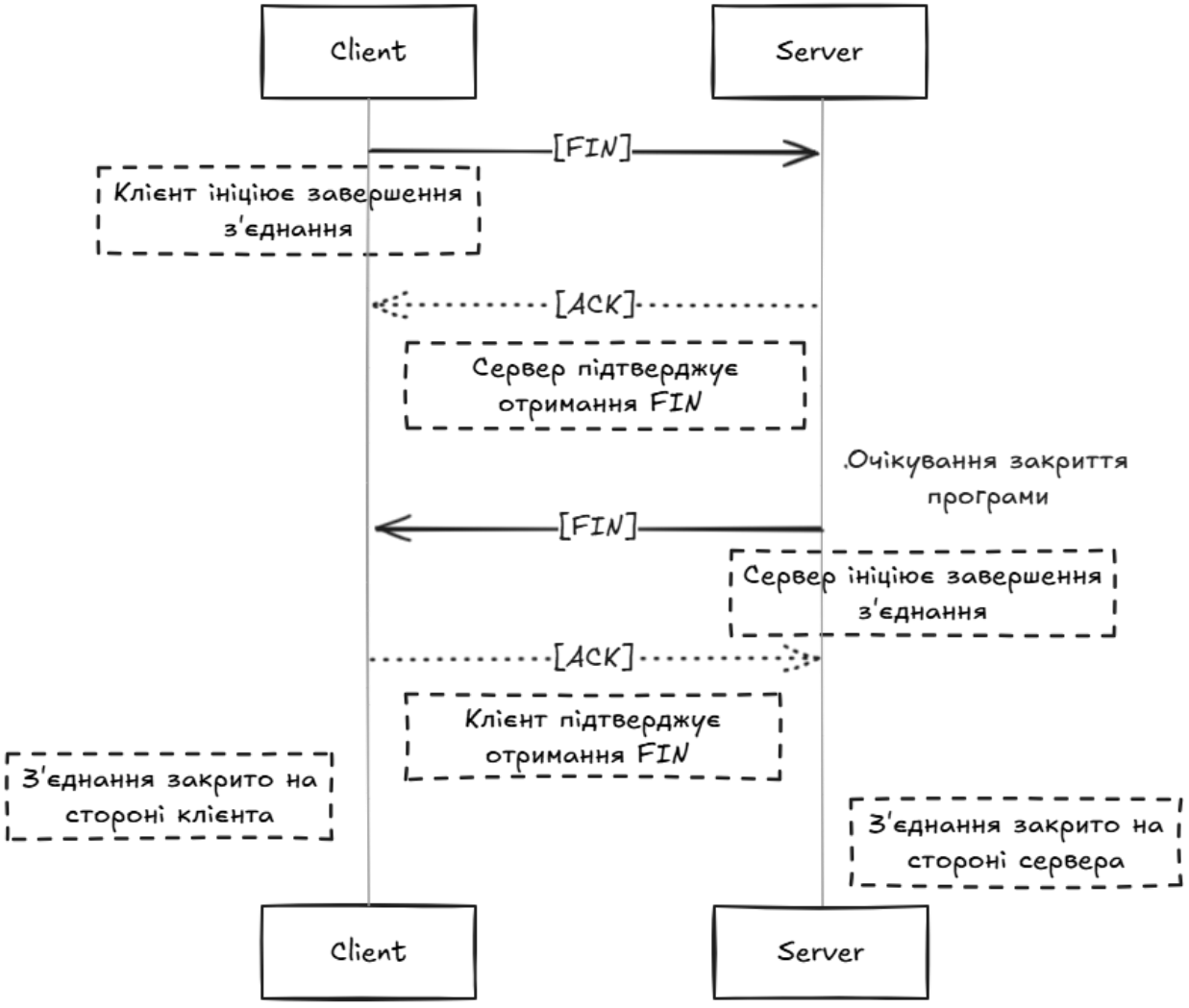

Рис. 1.4. Алгоритм завершення TCP з'єднання

Завдяки своїй структурі та вбудованим механізмам контролю, TCP залишається основним протоколом транспортного рівня для програм, що потребують точної, надійної та послідовної доставки інформації.

Завдання

1. Відділу виділено діапазон IP адрес у мережі 192.168.55.0. Необхідно в рамках відділу організувати дві підмережі. Необхідно навести вид маски, яка використовуватиметься в даному випадку.
2. Необхідно до таблиці маршрутизації додати адресу шлюзу 157.48.50.1 для доступу до мережі 160.50.0.0
3. Пояснити, чому найбільший номер мережі класу B дорівнює 191.255.0.0.
4. За заданою IP-адресою хоста 129.64.134.5 вказати номер мережі та номер вузла, якщо:
    1) маска 255.255.0.0
    2) маска 255. 255.128.0
5. Визначити фізичну адресу інтерфейсу комп'ютера.
6. Встановити статичну відповідність між IP та фізичною адресою сусідньої (будь-якої доступної) машини в таблиці ARP.
7. Визначити поточні підключення (за допомогою TCP/IP) до робочого місця.
8. Визначити список протоколів, що використовуються.
9. Привести формат команди для визначення доступності хоста 11.15.159.88, якщо відомо, що час відгуку становить не більше 200мс.
10. Для доступу до мережі 130.15.0.0 використовується шлюз 172.16.9.2, щоб привести команду для зміни таблиці маршрутизації.
11. Для доступу до мережі 195.15.46.0 використовуються шлюзи 172.16.9.2 та 172.16.9.9, навести команду зміни таблиці маршрутизації з урахуванням того, що шлюз 172.16.9.9 є додатковим.
12. Використовуючи запропонований аналізатор протоколів:
    1. Визначити хости (IP-адреси та протоколи, порти), що взаємодіють з робочим місцем.
    2. Визначити формат Ethernet-кадру.
    3. Визначити формат пакету TCP.
    4. Визначити формат пакета UDP.
    5. Визначити формат IP-пакета.
    6. Налаштувати аналізатор протоколів на перехоплення вихідного TCP трафіку на робочому місці.
    7. Налаштувати аналізатор протоколів на перехоплення всього трафіку на сервер.

Перелік документів для включення до звіту ІР № 1

- титульний лист роботи
- номер та формулювання питання;
- команда, яка використовується для відповіді на питання;
- скріншот результату роботи команди.

Питання для самоконтролю

1. Які існують класи IP-адрес? Чим вони відрізняються?
2. Що таке маска підмережі? Як її використовують для розділення мережі?
3. Яким чином можна поділити мережу класу C на 4 підмережі? Які маски при цьому застосовуються?
4. Які діапазони IP-адрес зарезервовано для приватного використання?
5. Для чого використовується таблиця маршрутизації? Якими командами можна її переглядати або змінювати?
6. Як працює команда ping і для чого вона застосовується?
7. У чому полягає різниця між командами tracert та route?
8. Що показує команда netstat? Яку інформацію про з’єднання вона дозволяє отримати?
9. Яка команда дозволяє призначити статичну відповідність IP та MAC-адрес?
10. Які основні функції виконує протокол TCP у моделі OSI?
11. Які поля містить заголовок TCP-сегмента? Яке їхнє призначення?
12. У чому полягає принцип встановлення та завершення TCP-з'єднання?
13. Що таке «вікно» у TCP та як воно впливає на продуктивність з’єднання?
14. Чим відрізняється TCP від UDP при передачі даних?

Індивідуальна робота № 2
СТВОРЕННЯ TCP ЕХО-СЕРВЕРА З ВИКОРИСТАННЯМ БІБЛІОТЕКИ WinSocket.

Теоретичний матеріал

Як відомо, в контексті стеку протоколів TCP/IP ***сокет*** – це засіб встановлення з'єднання між прикладною програмою та портом локального вузла мережі, що забезпечує обмін даними через мережеве середовище. Сокет є абстрактним поняттям.

Інтерфейс сокетів (socket API) – це набір системних викликів або бібліотечних функцій, що реалізуються, зокрема, мовою програмування С. Всі функції сокетів у середовищі Windows реалізовані в динамічній бібліотеці WS2_32.dll, а для успішної компіляції програми з використанням WinSock необхідно підключити відповідну бібліотеку *Ws2_32.lib*.

Socket-інтерфейс включає кілька груп функцій, однак у контексті створення мережевих застосунків у моделі «клієнт–сервер», які використовують встановлення з'єднання між учасниками взаємодії, зазвичай достатньо обмеженої підмножини цих функцій. Розгляд саме цієї підмножини дозволяє сконцентрувати увагу на базових принципах обміну даними на транспортному рівні мережі.

Розробка додатків з використанням сокетів починається з підключення *Ws2_32.lib*. У меню *Project* виберіть пункт *settings*, а там вкажіть розділ *Link*, Ws2_32.lib можна ввести в полі *Library modules* або *project options*. Не забудьте встановити `#include <winsock2.h>`

1. Створення сервера

Всі процеси (додатки або бібліотеки DLL), що викликають функції Winsock, повинні ініціалізувати використання бібліотеки DLL сокетів Windows перед викликом інших функцій *Winsock*. Це також гарантує, що Winsock підтримується в системі.

Перш ніж скористатися функцією socket необхідно проініціалізувати процес бібліотеки *WS2_32.dll* викликавши функцію ***WSAStartup***, наприклад:

1. Створити об'єкт WSADATA з ім'ям wsaData.

2. Викликати WSAStartup та повернути його значення у вигляді цілого числа. А також потрібно перевірити наявність помилок.

```
WSADATA wsaData;
int iResult;

// Initialize Winsock
```

```
iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
if (iResult != 0) {
      printf("WSAStartup failed: %d\n", iResult);
      return 1;}
```

Функція WSAStartup викликається для запуску використання WS2_32.dll.

Структура WSADATA містить інформацію про реалізацію сокетів Windows. Параметр MAKEWORD(2,2) WSAStartup запитує версію 2.2 Winsock в системі і задає передану версію як найвищу версію Windows Sockets підтримки, яку може використовувати об'єкт, що викликає.

Тепер переходимо до створення сокету. Після ініціалізації об'єкт SOCKET має бути створений для використання сервером.

Створення сокету для сервера

1. Функція *getaddrinfo* використовується для визначення значень структури sockaddr :

- AF_INET використовується для вказівки сімейства IPv4 адрес.
- SOCK_STREAM використовується для вказівки потоку сокету.
- IPPROTO_TCP використовується для вказівки протоколу TCP.
- AI_PASSIVE прапор вказує, що об'єкт, що викликає, має намір використовувати повертається структуру адреси сокета у виклику функції прив'язки . Якщо для функції *getaddrinfo* встановлено прапорець AI_PASSIVE, а для функції *getaddrinfo* встановлено значення NULL, для ip-адреси в структурі адрес сокета встановлюється значення INADDR_ANY для IPv4-адрес або IN6ADDR_ANY_INIT для IPv6-адрес.
- 27015 – це номер порту, пов'язаний із сервером, до якого підключатиметься клієнт.

Структура addrinfo використовується функцією getaddrinfo.

```
#define DEFAULT_PORT "27015"

struct addrinfo *result = NULL, *ptr = NULL, hints;

ZeroMemory(&hints, sizeof (hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
hints.ai_flags = AI_PASSIVE;

// Resolve the local address and port to be used by the server
iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
if (iResult != 0) {
      printf("getaddrinfo failed: %d\n", iResult);
      WSACleanup();
      return 1;
      }
```

2. Далі необхідно створити об'єкт SOCKET під назвою ***ListenSocket***. Цей об’єкт використовується для прослуховування сервером клієнтських підключень.

```
SOCKET ListenSocket = INVALID_SOCKET;
```

3. Наступним кроком є виклик функції сокета та повернення її значення до змінної ***ListenSocke***t.

Для цієї серверної програми рекомендується використовувати першу IP-адресу, яка повертається викликом ***getaddrinfo***, яка відповідає сімейству адрес, типу сокету та протоколу, вказаному у параметрі підказок. У наведеному коді запитується сокет потоку TCP для IPv4 із сімейством адрес IPv4, типом сокету ***SOCK_STREAM*** і протоколом ***IPPROTO_TCP***. Тому IPv4-адреса запитується для ***ListenSocket***.

```
// Create a SOCKET for the server to listen for client connections

      ListenSocket = socket(result->ai_family, result->ai_socktype,
result->ai_protocol);
```

4. Наступним кроком необхідно перевірити наявність помилок, щоб переконатися, що сокет є допустимим сокетом. Слід зауважити, що перевірку наявності помилок при використанні сокетів та функції, що з ними пов’язані, рекомендується робити кожного разу, при використанні відповідних програмних конструкцій.

```
if (ListenSocket == INVALID_SOCKET) {
      printf("Error at socket(): %ld\n", WSAGetLastError());
      freeaddrinfo(result);
      WSACleanup();
      return 1;
      }
```

Щоб сервер приймав клієнтські підключення, він повинен бути прив'язаний до адреси мережі в системі. У наступному коді показано, як пов’язати сокет, який вже було створено, з IP-адресою та портом. Без функції ***bind*** сервер не буде знати, на який порт і адресу мережі приймати вхідні з’єднання.

Прив’язка сокета.

Структура ***sockaddr*** містить відомості про сімейство адрес, IP-адресу та номер порту. Для прив’язки використовується відповідна функція ***bind***, в якості вхідних параметрів якої передається дескриптор сокета ***ListenSocket***., створений раніше. Це сокет, який буде слухати вхідні з’єднання. Також передається вказівник на структуру ***sockaddr***, яка містить IP-адресу і порт, що треба прив’язати. Ця структура зазвичай заповнюється функцією ***getaddrinfo***. І останнім параметром є розмір структури ***sockaddr***, який теж можна отримати за допомогою функції ***getaddrinfo***. Як зазначалося вище, на цьому етапі також потрібно передбачити перевірку на помилки.

```
// Setup the TCP listening socket
iResult = bind( ListenSocket, result->ai_addr, (int)result-
>ai_addrlen);
if (iResult == SOCKET_ERROR) {
      printf("bind failed with error: %d\n", WSAGetLastError());
      freeaddrinfo(result);
      closesocket(ListenSocket);
      WSACleanup();
      return 1;
      }
```

Після виклику функції прив'язки відомості про адресу, що повертаються функцією ***getaddrinfo***, більше не потрібні. Функція ***freeaddrinfo*** викликається для звільнення пам'яті, виділеної функцією ***getaddrinfo*** для цих відомостей про адресу.

Після прив'язки сокету до IP-адреси та порту в системі сервер повинен прослуховувати цю IP-адресу та порт для вхідних запитів на підключення.

Прослуховування сокету

Для забезпечення прослуховування сервером відповідної адреси використовується функція ***listen***, вхідними параметрами якої є створений

сокет і максимальна довжина черги очікування з’єднань. У наведеному нижче фрагменті коду для параметра черги очікування встановлено значення ***SOMAXCONN***. Це значення є спеціальною константою, яка вказує постачальнику Winsock для цього сокету дозволити максимально допустиму кількість очікуваних підключень у черзі. Зазвичай це 128, але значення може змінюватись залежно від системи

```
if ( listen( ListenSocket, SOMAXCONN ) == SOCKET_ERROR ) {
      printf( "Listen failed with error: %ld\n",
      WSAGetLastError() );
      closesocket(ListenSocket);
      WSACleanup();
      return 1;
      }
```

Коли сокет прослуховує підключення, програма повинна обробляти запити на підключення цього сокету.

Прийняття підключення до сокету

Для забезпечення можливості прийняття запиту від клієнта на встановлення з’єднання (підключення) необхідно створити тимчасовий об'єкт ***SOCKET*** з ім'ям ***ClientSocket***.

```
SOCKET ClientSocket;
```

Як правило, серверна програма призначена для прослуховування підключень від кількох клієнтів. Для високопродуктивних серверів часто використовують кілька потоків для обробки кількох клієнтських підключень.

Існує кілька різних методів програмування за допомогою ***Winsock***, які можна використовувати для прослуховування кількох під'єднань клієнтів. Одним із способів програмування є створення нескінченного циклу, який перевіряє запити на підключення за допомогою функції прослуховування. Якщо запит на підключення виникає, програма викликає функцію ***accept, AcceptEx*** або ***WSAAccept*** і передає роботу іншому потоку для обробки запиту. Можливі інші методи програмування.

```
ClientSocket = INVALID_SOCKET;

// Accept a client socket
ClientSocket = accept(ListenSocket, NULL, NULL);
if (ClientSocket == INVALID_SOCKET) {
      printf("accept failed: %d\n", WSAGetLastError());
      closesocket(ListenSocket);
      WSACleanup();
      return 1;
      }
```

Зауважте, що цей базовий приклад дуже простий і не використовує кілька потоків. Приклад також просто прослуховує та приймає лише одне підключення.

Після прийняття клієнтського підключення серверна програма зазвичай передає прийнятий сокет клієнта (змінну ***ClientSocket*** у наведеному вище фрагменті коду) робочому потоку або порту завершення введення-виводу і продовжує приймати додаткові підключення. У цьому базовому прикладі сервер продовжує перехід до наступного кроку, оскільки працює тільки з одним з'єднанням.

Отримання та відправлення даних у сокет

Для передачі даних між клієнтом і сервером використовуються функції ***recv*** – отримання даних та ***send*** – відправка даних. Використання цих функцій є однаковим як для серверної, так і для клієнтської програми.

У наступному фрагменті коду показано функції ***recv*** та ***send*** сервера. Функції ***send*** та ***recv*** повертають ціле значення числа відправлених або отриманих байтів відповідно або помилку. Кожна функція приймає такі параметри: активний сокет, буфер **char** для збереження отриманих даних чи для даних, які потрібно передати, кількість байтів для відправлення або отримання, а також прапори управління прийомом/передачею.

```
#define DEFAULT_BUFLEN 512

char recvbuf[DEFAULT_BUFLEN];
int iResult, iSendResult;
int recvbuflen = DEFAULT_BUFLEN;

// Receive until the peer shuts down the connection
do {

      iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
      if (iResult > 0) {
            printf("Bytes received: %d\n", iResult);

      // Echo the buffer back to the sender
            iSendResult = send(ClientSocket, recvbuf, iResult, 0);
            if (iSendResult == SOCKET_ERROR) {
                  printf("send failed: %d\n", WSAGetLastError());
                  closesocket(ClientSocket);
                  WSACleanup();
                  return 1;
            }
            printf("Bytes sent: %d\n", iSendResult);
      } else if (iResult == 0)
            printf("Connection closing...\n");
```

```
        else {

            printf("recv failed: %d\n", WSAGetLastError());
            closesocket(ClientSocket);
            WSACleanup();
            return 1;
        }

    } while (iResult > 0);
```

Завершивши отримання даних від клієнта та відправляючи дані назад клієнту, сервер відключається від клієнта та завершує роботу сокету.

Вимкнення та завершення роботи сокету

1. Після надсилання даних клієнту на сервері можна викликати функцію ***shutdown***, вказавши SD_SEND для завершення надсилання сокета. Це дозволяє клієнту звільнити деякі ресурси цього сокету. При цьому серверна програма, як і раніше, може отримувати дані в сокеті.

```
    // shutdown the send half of the connection since no more data
    will be sent

    iResult = shutdown(ClientSocket, SD_SEND);

    if (iResult == SOCKET_ERROR) {

        printf("shutdown failed: %d\n", WSAGetLastError());

        closesocket(ClientSocket);

        WSACleanup();

        return 1;}
```

Коли клієнтська програма отримує дані, функція ***closesocket*** викликається закриття сокета. Після завершення роботи клієнтської програми за допомогою бібліотеки DLL сокетів Windows викликається функція ***WSACleanup*** для звільнення ресурсів.

```
    // cleanup
    closesocket(ClientSocket);
    WSACleanup();

        return 0;
```

2. Створення клієнта

Так само, як і для сервера, процеси, що викликають функції Winsock, повинні ініціалізувати використання відповідної бібліотеки перед викликом

інших функцій. Ініціалізація процесу бібліотеки WS2_32.dll відбувається так само, як і для сервера. Після ініціалізації об'єкт SOCKET має бути створений для користування клієнтом.

Створення сокету

1. Необхідно оголосити об'єкт ***addrinfo*** , який містить структуру ***sockaddr*** та проініціалізувати значення відповідні значення. Для цієї програми сімейство адрес Інтернету не вказано. Це зроблено для того, щоб можна було повернути адресу IPv6 або IPv4. Програма запитує тип сокету як сокет потоку для протоколу TCP.

```
struct addrinfo *result = NULL,
*ptr = NULL,
hints;
ZeroMemory( &hints, sizeof(hints) );
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
```

2. Після ініціалізації структури ***addrinfo*** необхідно викликати функцію ***getaddrinfo*** , яка запитує IP-адресу для імені сервера, переданого в командному рядку. TCP-порт на сервері, до якого буде підключатися клієнт, визначається DEFAULT_PORT як 27015 у цьому прикладі. Функція ***getaddrinfo*** повертає значення цілого числа, яке перевіряється на наявність помилок.

```
#define DEFAULT_PORT "27015"

// Resolve the server address and port
iResult = getaddrinfo(argv[1], DEFAULT_PORT, &hints, &result);
if (iResult != 0) {
      printf("getaddrinfo failed: %d\n", iResult);
      WSACleanup();
      return 1;
            }
```

3. Наступним кроком потрібно створити об'єкт SOCKET під назвою ***ConnectSocket***.

```
SOCKET ConnectSocket = INVALID_SOCKET;
```

4. Далі переходимо до виклику функції ***socket*** **,** результат роботи якої повертається у змінну ***ConnectSocket***. Для цієї програми використовується перша IP-адресу, повернена викликом **getaddrinfo** , яка відповідала сімейству адрес, типу сокету та протоколу, вказаному у параметрі підказок . У нижче наведеному прикладі було зазначено сокет TCP-потоку з типом сокету SOCK_STREAM та протоколом IPPROTO_TCP. Сімейство адрес не було

вказано (AF_UNSPEC), тому повернена IP-адреса може бути IPv6 або IPv4-адресою сервера.

```
// Attempt to connect to the first address returned by
// the call to getaddrinfo
ptr=result;
// Create a SOCKET for connecting to server

ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, x ptr-
>ai_protocol);
```

5. Далі переходимо до перевірки на наявність помилок, щоб переконатися, що сокет є допустимим сокетом.

```
if (ConnectSocket == INVALID_SOCKET) {
      printf("Error at socket(): %ld\n", WSAGetLastError());
      freeaddrinfo(result);
      WSACleanup();
      return 1;
      }
```

Параметри, що передаються функції ***socket***, можна змінити для різних реалізацій.

Виявлення помилок є ключовою частиною успішного мережного коду. Якщо виклик сокета завершується збоєм, він повертає значення INVALID_SOCKET. Оператор ***if*** у попередньому фрагменті коду використовується для перехоплення помилок, які могли б виникнути при створенні сокету. *WSAGetLastError* повертає номер помилки, пов'язаний з останньою помилкою.

Щоб клієнт мав змогу взаємодіяти у мережі, він має підключитись до сервера.

Підключення до сервера

Функція ***connect*** використовується на клієнтському боці для встановлення TCP-з’єднання з віддаленим сервером. В якості параметрів функція ***connect*** приймає дескриптор сокета, вказівник на структуру ***sockaddr***, яка містить IP-адресу і порт сервера та розмір цієї структури. При успішному виконання функції ***connect*** повертається значення **0**, а у разі виникнення помилки повертається значення ***SOCKET_ERROR (-1).***

```
// Connect to server.
iResult = connect( ConnectSocket, ptr->ai_addr, (int)ptr-
>ai_addrlen);
if (iResult == SOCKET_ERROR) {
      closesocket(ConnectSocket);
      ConnectSocket = INVALID_SOCKET;
```

```
    }

// Should really try the next address returned by getaddrinfo
// if the connect call failed
// But for this simple example we just free the resources
// returned by getaddrinfo and print an error message

freeaddrinfo(result);

if (ConnectSocket == INVALID_SOCKET) {
    printf("Unable to connect to server!\n");
    WSACleanup();
    return 1;
    }
```

Функція ***getaddrinfo*** використовується для визначення значень у структурі **sockaddr.** У цьому прикладі перша IP-адреса, що повертається функцією **getaddrinfo**, використовується для вказівки структури ***sockaddr***, переданої у з'єднання. Якщо під час виклику підключення не вдається отримати першу IP-адресу, спробуйте наступну структуру ***addrinfo*** у зв'язаному списку, поверненому функцією ***getaddrinfo*** .

Відомості, вказані в структурі sockaddr, включають:

- IP-адресу сервера, до якого клієнт пробує підключитись;
- номер порту на сервері, до якого підключатиметься клієнт. Цей порт був вказаний як порт 27015, коли клієнт викликав функцію ***getaddrinfo***.

Функції відправки та отримання даних для сервера та для клієнта аналогічні.

Завдання

Розробити дві програми: TCP-клієнт і TCP exo-сервер з використанням бібліотеки WinSock мовою програмування C. Забезпечити між ними двосторонній обмін повідомленнями через протокол TCP.

1. Серверна частина (TCP exo-сервер):

- Повинен прослуховувати заданий порт на локальному комп’ютері.
- Очікує на вхідні підключення від одного або кількох клієнтів.
- Після встановлення з’єднання:
    - приймає від клієнта текстовий рядок (повідомлення),
    - виводить цей рядок на екран сервера разом з IP-адресою клієнта, який його надіслав,

- надсилає отримане повідомлення назад клієнту без змін.

- Після завершення обміну даними з клієнтом з'єднання може бути завершене або продовжене (залежно від реалізації).

2. Клієнтська частина (TCP-клієнт):

- Повинна встановити TCP-з’єднання із сервером (на локальній або вказаній IP-адресі).
- Надсилає введений користувачем текстовий рядок до сервера.
- Отримує відповідь від сервера та виводить її на екран клієнта.
- Клієнт має завершити роботу після надсилання та отримання одного або кількох повідомлень.

Перелік документів для включення до звіту ІР № 2

- титульний лист;
- постановка задачі;
- код розроблених програм з коментарями;
- скріншоти результатів роботи програми.

Питання для самоконтролю

1. У чому полягає різниця між сервером і клієнтом у TCP-з’єднанні?
2. Які етапи включає створення TCP-сервера?
3. Які етапи реалізації TCP-клієнта?
4. Для чого використовується функція WSAStartup()?
5. Для чого використовується функція bind()? Які її параметри?
6. Яку роль виконує функція listen()?
7. Як працює функція accept()? Яке її призначення у сервері?
8. Що таке connect() і на якому боці (клієнт чи сервер) вона використовується?
9. Що роблять функції send() і recv()? У чому відмінність між ними?
10. Що означає значення, яке повертає recv() рівне 0?
11. Як визначити, що з’єднання було успішно встановлено, або що виникла помилка?
12. Які дії потрібно виконати перед використанням WinSock і після завершення роботи з нею?
13. Як організувати нескінченний цикл обробки клієнтів на сервері?

14. У чому полягає обробка кількох клієнтів: що потрібно реалізувати, щоб сервер міг працювати з кількома клієнтами одночасно?

Індивідуальна робота № 3
РОЗРОБКА HTTP-СЕРВЕРА.

Теоретичний матеріал

HTTP (HyperText Transfer Protocol)

Це протокол передачі гіпертексту, який використовується в глобальній мережі Інтернет для обміну інформацією між веб-клієнтами (наприклад, браузерами) і веб-серверами. Це протокол прикладного рівня моделі OSI, побудований за принципом клієнт-серверної архітектури.

Основна роль HTTP полягає в тому, щоб забезпечити механізм надсилання запитів клієнтом і отримання відповідей від сервера. При цьому HTTP дозволяє передавати:

- HTML-документи
- CSS- і JavaScript-файли
- зображення, відео, аудіо
- дані у форматі JSON, XML тощо.

HTTP працює на прикладному рівні (7 рівень) моделі OSI. Він спирається на протоколи нижчих рівнів, зокрема:

- TCP – забезпечує надійну передачу даних (4 рівень)
- IP – маршрутизація пакетів у мережі (3 рівень)

Таблиця 3.1. Рівні моделі OSI

Рівень моделі OSI		Приклади протоколів
7	Прикладний (Application)	**HTTP, FTP, SMTP, DNS**
6	Представлення (Presentation)	ASCII, JPEG, Unicode
5	Сеансовий (Session)	NetBIOS, SSL
4	Транспортний (Transport)	**TCP, UDP**
3	Мережевий (Network)	**IPv4, Ipv6, ICMP**
2	Канальний (Data Link)	**Ethernet, WiFi, ARP**
1	Фізичний (Physical)	**Кабелі, радіо, мережеві інтерфейси**

Архітектура HTTP протоколу

HTTP працює за моделлю **«запит-відповідь»**. Клієнт (наприклад, браузер) надсилає **HTTP-запит**, а сервер повертає **HTTP-відповідь**.

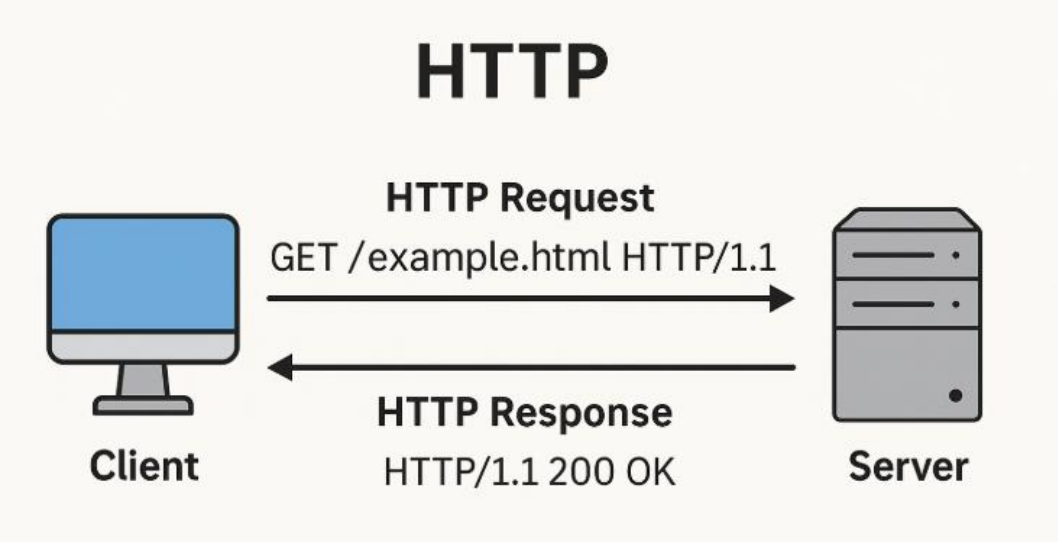

Рис 3.1. Клієнт-серверна архітектура протоколу HTTP

HTTP запит (HTTP Request)

HTTP-запит – це повідомлення, яке надсилає клієнт (наприклад, веб-браузер) на веб-сервер для отримання певного ресурсу або взаємодії з сервером. Кожен запит формує початок циклу «клієнт–сервер», у якому сервер повинен обробити запит і надіслати відповідь.

HTTP-запит – це текстове повідомлення, яке складається з ***трьох основних частин***:

- стартовий рядок або рядок стану (Request Line), який містить метод запиту, шлях до ресурсу та версію протоколу;
- поля заголовків (Headers);
- тіло запиту (Body)

Рис.3.2. Структура http-запиту

Методи HTTP-запитів

Методи HTTP вказують, яку дію клієнт хоче виконати щодо ресурсу на сервері. Існує п’ять основних методів та 4 додаткові.

Таблиця 3.2. Методи http запиту

Метод	Опис	Приклад HTTP-запиту
Основні методи		
GET	Отримання ресурсу з сервера. Не має тіла запиту. Найпоширеніший метод.	GET /images/logo.png HTTP/1.1 Host: example.com
POST	Надсилання даних на сервер (наприклад, форма). Дані передаються в тілі запиту.	POST /submit-form HTTP/1.1 Host: example.com Content-Type: application/x-www-form-urlencoded Content-Length: 27 name=Іван&email=ivan@mail.com
PUT	Завантаження або створення ресурсу. Тіло містить повну інформацію про ресурс.	PUT /users/ivan HTTP/1.1 Host: example.com Content-Type: application/json Content-Length: 38 {"name": "Іван", "email": "ivan@mail.com"}
DELETE	Видалення ресурсу на сервері.	DELETE /users/ivan HTTP/1.1 Host: example.com
HEAD	Як GET, але без тіла відповіді. Використовується для перевірки наявності ресурсу або метаданих.	HEAD /index.html HTTP/1.1 Host: example.com
Додаткові методи		
OPTIONS	Отримання інформації про підтримувані сервером методи для вказаного ресурсу.	OPTIONS /api/data HTTP/1.1 Host: example.com
PATCH	Часткове оновлення існуючого ресурсу.	PATCH /users/ivan HTTP/1.1 Host: example.com Content-Type: application/json Content-Length: 22 {"email": "new@mail.com"}
CONNECT	Встановлення TCP-тунелю, зазвичай використовується для **HTTPS** через проксі.	CONNECT www.example.com:443 HTTP/1.1 Host: www.example.com
TRACE	Тестовий запит: сервер повертає клієнту запит у тому вигляді, як його отримав (для діагностики).	TRACE /test HTTP/1.1 Host: example.com

Шлях до ресурсу

У HTTP-запиті шлях до ресурсу вказує, який саме об'єкт на сервері клієнт хоче отримати, змінити чи видалити. Цей шлях починається після доменного імені (або IP-адреси) й часто відображає фізичну або логічну структуру каталогів на сервері, хоча іноді сервер може «перехоплювати» шлях програмно (наприклад, у фреймворках).

У запиті GET /index.html HTTP/1.1 шлях до ресурсу — це /index.html.

Шлях до ресурсу поділяється на 4 типи, які наведені у таблиці нижче

Таблиця 3.3. Типи шляхів до ресурсу

Тип шляху	Приклад	Опис
Абсолютний	/docs/report.html	Вказує повний шлях від кореня сервера
Відносний	report.html	Відносно поточної директорії (найчастіше використовується в HTML)
Із параметрами	/search?q=chatgpt	Містить параметри запиту (після ?)
Із якірцем	/about#contact	Вказує на конкретний фрагмент сторінки (використовується тільки на стороні клієнта)

В протоколі HTTP, коли говорять про шлях до ресурсу, використовують поняття ***URI*** та/або ***URL.***

URI (Uniform Resource Identifier) **–** це уніфікований ідентифікатор ресурсу, загальний термін, який ідентифікує ресурс у просторі імен, незалежно від того, чи можна його знайти фізично.

URI ≈ імена + адреси.

URL (Uniform Resource Locator) **–** це підтип URI, який вказує точне розташування ресурсу та спосіб доступу до нього (через протокол).

URL = URI + адреса + протокол доступу.

URL обов’язково містить схему (http, https, ftp, mailto тощо), адресу (домен або IP) та шлях до ресурсу. Також опційно може містити порт, параметри, якірці, авторизацію тощо. Наприклад:

https://example.com/docs/index.html *Визначає розташування і схему*
http://example.com/search?q=network *Містить протокол, домен і параметри*

HTTP відповідь (HTTP Response)

HTTP-відповідь – це повідомлення, яке веб-сервер надсилає у відповідь на HTTP-запит від клієнта (наприклад, браузера). Вона містить інформацію про те, як сервер обробив запит, а також – за потреби – сам запитуваний ресурс (наприклад, HTML-документ, зображення тощо).

HTTP-відповідь, як і запит, складається з трьох основних частин:

- Стартовий рядок або рядок стану (Status Line), який містить версії протоколу (наприклад, HTTP/1.1), коду стану (наприклад, 200) та пояснення до коду (OK).
- HTTP-заголовки (Headers), які містять службову інформацію про тип вмісту (Content-Type), довжину тіла відповіді (Content-Length), дату, сервер, кешування тощо.
- Тіло відповіді (Body) - вміст, який сервер надсилає клієнту — HTML-код сторінки, JSON-дані, зображення, PDF, текст тощо. Для деяких методів (HEAD) тіло відсутнє.

Рис.3.3. Структура http-відповіді

Коди стану відповіді

HTTP використовує тризначні коди стану, які вказують на результат обробки запиту. Перший символ коду визначає загальний клас:

Таблиця 3.4. Класи кодів стану

Клас	Діапазон	Опис
1xx	100–199	Інформаційні
2xx	200–299	Успішні
3xx	300–399	Перенаправлення
4xx	400–499	Помилки клієнта
5xx	500–599	Помилки сервера

Розглянемо детальніше основні коди станів, які використовуються у протоколі HTTP

Таблиця 3.5. Основні коди стану http-відповіді

2xx — Успішні відповіді		
Код	**Опис**	**Приклад відповіді**
200 OK	Запит оброблено успішно	HTTP/1.1 200 OKContent-Type: text/html<html>...</html>
201 Created	Ресурс створено	HTTP/1.1 201 CreatedLocation: /new-resource
202 Accepted	Запит прийнято для обробки, але ще не виконано	HTTP/1.1 202 Accepted
204 No Content	Запит виконано, але без вмісту	HTTP/1.1 204 No Content
3xx — Перенаправлення		
301 Moved Permanently	Ресурс переміщено назавжди	HTTP/1.1 301 Moved PermanentlyLocation: https://newsite.com
302 Found	Тимчасове перенаправлення	HTTP/1.1 302 FoundLocation: /temporary-page
304 Not Modified	Ресурс не змінився (для кешування)	HTTP/1.1 304 Not Modified
4xx — Помилки клієнта		
400 Bad Request	Невірний запит (синтаксична помилка)	HTTP/1.1 400 Bad Request
401 Unauthorized	Потрібна авторизація	HTTP/1.1 401 Unauthorized``WWW-Authenticate: Basic realm="Access"`
403 Forbidden	Доступ до ресурсу заборонено	HTTP/1.1 403 Forbidden
404 Not Found	Ресурс не знайдено	HTTP/1.1 404 Not Found <html> 404 Page not found </html>
405 Method Not Allowed	Метод не підтримується для цього ресурсу	HTTP/1.1 405 Method Not Allowed
408 Request Timeout	Тайм-аут запиту	HTTP/1.1 408 Request Timeout
5xx — Помилки сервера		
500 Internal Server Error	Загальна помилка сервера	HTTP/1.1 500 Internal Server Error
501 Not Implemented	Метод не реалізований на сервері	HTTP/1.1 501 Not Implemented
502 Bad Gateway	Невірна відповідь проміжного шлюзу	HTTP/1.1 502 Bad Gateway
503 Service Unavailable	Сервер тимчасово недоступний	HTTP/1.1 503 Service Unavailable
504 Gateway Timeout	Тайм-аут шлюзу	HTTP/1.1 504 Gateway Timeout

Завдання

Розробити HTTP-сервер, який прослуховує 80 порт (або дублюючий порт 8080) і призначений для обробки HTTP-запитів з методом GET та надсилання HTTP-відповідей стандартному браузеру. Для реалізації серверу можна використовувати мови програмування Java, Python або C#.

HTML-сторінки, які повертає HTTP-сервер, повинні бути збережені в конкретному каталозі, який може мати вкладену структуру. Якщо в рядку браузера не вказати конкретну сторінку, сервер повинен повернути сторінку *index.html*, що відповідає даному каталогу. Якщо сторінка, яку запитує користувач, недоступна або не існує, браузер повинен відображати сторінку з помилкою. Також необхідно передбачити можливість обробляти запити на отримання зображень. Тобто при формуванні відповіді обов'язкова наявність HTTP-заголовків Content-Type та Content-Length.

Всі HTML-сторінки, включаючи сторінку з помилкою, студент розробляє самостійно. Вони повинні бути простими за структурою, обов'язково містити теги <head>, <title>, <body>.

Структуру каталогів, назви та кількість файлів у каталогах студент обирає самостійно. Можна використовувати одну сторінку *index.html* для відповіді на всі запити до каталогів, або створити окремі сторінки для кожного каталогу.

Перехід по HTML-сторінками за допомогою гіперпосилань не зараховується як успішне виконання роботи. Всі відображені користувачу сторінки повинні бути результатом обробки конкретного GET-запиту.

Приклад роботи:

При переході за адресою http://127.0.0.1/ браузер повинен відобразити файл *index.html* із кореневої папки проєкту.

При переході за адресою http://127.0.0.1/myFolder/title.html браузер повинен відобразити файл *title.html* із папки myFolder.

При переході за адресою http://127.0.0.1/somePage.html , якщо сторінки somePage.html не існує, браузер повинен відобразити файл *errorPage.html* з повідомлення про помилку «404 Not Found».

Перелік документів для включення до звіту ІР № 3

- титульний лист;
- постановка задачі;
- код розробленого класу з коментарями;
- скріншот структури каталогу з файлами серверу (HTML-сторінками та додатковими файлами);
- скріншоти результатів роботи програми.

Питання для самоконтролю

1. Що таке HTTP? До якого рівня моделі OSI належить цей протокол?
2. Яка структура HTTP-запиту? Які поля обов’язкові?
3. Які основні методи HTTP-запиту ви знаєте? У чому полягають відмінності між ними?
4. Яка відмінність між методами GET, POST, PUT та DELETE з точки зору взаємодії із сервером?
5. Яка структура HTTP-відповіді?
6. Яке значення мають коди стану у HTTP-відповіді? Наведіть приклади кодів кожної категорії (1xx – 5xx).
7. Що таке 404 Not Found та у яких випадках цей код повертається сервером?
8. Що таке заголовки (headers) в HTTP? Наведіть приклади заголовків запиту та відповіді.
9. Який зміст поняття «тіло запиту» і які методи його використовують?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна література

1. Лосев Ю. І., Руккас К. М., Шматков С. І. Комп’ютерні мережі : навч. посіб. / За ред. Ю. І. Лосева. Харків : ХНУ імені В. Н. Каразіна, 2013. 248 с.
2. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 8th ed. Boston : Pearson, 2020. 864 p.
3. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Boston : Pearson, 2011. 960 p.
4. Peterson L. L., Davie B. S. Computer Networks: A Systems Approach. 6th ed. Amsterdam : Morgan Kaufmann, 2021. 848 p.
5. Stevens W. R. TCP/IP Illustrated. Volume 1: The Protocols. Boston : Addison-Wesley, 2011. 560 p.
6. RFC 9293 – Transmission Control Protocol (TCP) – об'єднане, оновлене визначення протоколу TCP [Електронний ресурс] / IETF. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc9293
7. RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1 [Електронний ресурс] / Internet Engineering Task Force. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc2616

Допоміжна література

1. Comer D. E. Computer Networks and Internets. 6th ed. Boston : Pearson, 2014. 672 p.
2. Van Winkle L. Hands-On Network Programming with C: Learn socket programming in C and write secure and optimized network code. Birmingham : Packt Publishing, 2019. 460 p.
3. Sanders C. Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems. 3rd ed. San Francisco : No Starch Press, 2017. 368 p.
4. Williams E. Learning HTTP/2: A Practical Guide for Beginners. Independently published, 2020. 125 p.
5. MSDN. Windows Sockets 2 (WinSock2) [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/windows/win32/WinSock/windows-sockets-start-page-2
6. Wireshark. Офіційний сайт аналізатора мережевих пакетів [Електронний ресурс]. – Режим доступу: https://www.wireshark.org/

ДОДАТКИ

Додаток № 1. Титульний лист індивідуальної роботи

Міністерство освіти і науки України

Харківський національний університет імені В. Н. Каразіна

Кафедра теоретичної та прикладної інформатики

Звіт з дисципліни
«Інформаційні мережі»
Індивідуальна робота № __
на тему «_____»

Виконав студент:_____

Група:_____

Необхідний термін здачі роботи:_____

Фактичний термін здачі роботи:_____

Кількість балів:_____

Харків рік

Електронне навчальне видання комбінованого використання
Можна використовувати в локальному та мережному режимі

Руккас Кирило Маркович
Леонова Наталія Ігорівна
Фролов В’ячеслав Вікторович
Носов Костянтин Валентинович

ІНФОРМАЦІЙНІ МЕРЕЖІ

Методичні рекомендації до виконання
індивідуальних (розрахунково-графічних) робіт для здобувачів вищої освіти
першого (бакалаврського) рівня факультету математики і інформатики

В авторській редакції

Підписано до розміщення 23.10.2025. Гарнітура Times New Roman.
Ум. друк. арк. 2,95. Обсяг 1,285 Мб. Зам. № 485/25.

Харківський національний університет імені В. Н. Каразіна,
61022, м. Харків, майдан Свободи, 4.
Свідоцтво суб’єкта видавничої справи ДК № 3367 від 13.01.2009
Видавництво ХНУ імені В. Н. Каразіна