Лабораторна робота № 5

Тема: Робота з мережевими вводом-виводом, сокети.

Мета: Навчитися розробляти програми керування мережними засобами операційних систем.

Обладнання та програмне забезпечення: ПК, операційна система Windows, середовище розробки Visual Studio.

Вказівки для самостійної підготовки

Під час підготовки необхідно повторити теоретичний матеріал:

1. Загальні принципи мережної підтримки.
2. Реалізація стека протоколів Інтернету.
3. Система імен DNS.
4. Програмний інтерфейс сокетів Берклі.
5. Архітектура мережної підтримки Windows.
6. Програмний інтерфейс Windows Sockets.

Теоретичні відомості

1 Загальні принципи мережної підтримки

Мережа - набір комп'ютерів або апаратних пристроїв (***вузлів***), пов'язаних між собою каналами зв'язку, які можуть передавати інформацію один одному. Мережа має конкретну фізичну структуру (***топологію***), усі вузли підключають до мережі із використанням апаратного забезпечення. Мережа об'єднує обмежену кількість вузлів.

Під ***інтернетом*** (з малої літери) розуміють сукупність мереж, які використовують один і той самий набір ***мережних протоколів*** — правил, що визначають формат даних для пересилання мережею. Фізична структура окремих мереж, які входять до складу інтернету, може різнитися. Такі різнорідні мережі пов'язують одну з одною ***маршрутизатори***, які переадресовують пакети з однієї мережі в іншу, залежно від їхньої адреси призначення і при цьому перетворюють пакети між форматами відповідних мереж. Маршрутизатори підтримують міжмережну взаємодію.

Інтернет (з великої літери)— це сукупність пов'язаних між собою інтернетів, відкритих для публічного доступу, які використовують визначений набір протоколів і охоплюють увесь світ.

1.1 Рівні мережної архітектури і мережні сервіси

Функції забезпечення зв'язку між вузлами є складними. Для спрощення їхньої реалізації використовують багаторівневий підхід - вертикальний розподіл мережних функцій і можливостей. Він дає змогу приховувати складність реалізації функцій зв'язку: кожен рівень приховує від вищих рівнів деталі реалізації своїх функцій та функцій, реалізованих на нижчих рівнях.

Мережний сервіс — це набір операцій, які надає рівень мережної архітектури для використання на вищих рівнях. Сервіси визначено як частину специфікації інтерфейсу рівня.

Розрізняють сервіси, ***орієнтовані на з'єднання*** і ***без з'єднань***, або ***дейтаграмні*** сервіси:

Сервіси, орієнтовані на з'єднання, реалізують три фази взаємодії із верхнім рівнем: встановлення з'єднання, передавання даних і розрив з'єднання. При цьому передавання даних на верхніх рівнях здійснюють у вигляді неперервного потоку байтів.

Дейтаграмні сервіси реалізують пересилання незалежних повідомлень, які можуть переміщатися за своїми маршрутами і приходити у пункт призначення в іншому порядку.

1.2 Мережні протоколи

Мережний протокол — це набір правил, що задають формат повідомлень, порядок обміну повідомленнями між сторонами та дії, необхідні під час передавання або прийому повідомлень.

Процес визначення адресата пакета за інформацією із його заголовків називають ***демультиплексуванням пакетів.***

Мережний протокол надає два інтерфейси:

Однорівневий, або ***інтерфейс протоколу*** призначений для взаємодії із реалізацією протоколу того самого рівня на віддаленому мережному вузлі. Це інтерфейс протоколу, що реалізує безпосереднє передавання даних на віддалений вузол. Такий інтерфейс забезпечують заголовком пакета, який доповнюють реалізацією цього протоколу перед передаванням пакета мережею.

Інтерфейс сервісу призначений для взаємодії із засобами вищого рівня; за його допомогою реалізують мережний сервіс.

Набір протоколів різного рівня, що забезпечують реалізацію певної мережної архітектури, називають ***стеком протоколів*** або ***набором протоколів***.

2 Реалізація стека протоколів Інтернету

Сукупність протоколів, які лежать в основі Інтернету, називають ***набором протоколів Інтернету*** або ***стеком протоколів TCP/IP*** за назвою двох основних протоколів.

2.1 Рівні мережної архітектури TCP/IP

Мережна архітектура TCP/IP має чотири рівні.

Канальний рівень відповідає за передавання кадру даних між будь-якими вузлами в мережах із типовою апаратною підтримкою або між двома сусідніми вузлами у будь-яких мережах. При цьому забезпечуються формування пакетів, корекція апаратних помилок, спільне використання каналів. На більш низькому рівні він забезпечує передавання бітів фізичними каналами, такими як коаксіальний кабель, кручена пара або оптоволоконний кабель (іноді для опису такої взаємодії виділяють окремий ***фізичний рівень***).

Хостом є вузол мережі, де використовують стек протоколів TCP/IP.

Мережним інтерфейсом є абстракція віртуального пристрою для зв'язку із мережею, яку надає програмне забезпечення канального рівня. Хост може мати декілька мережних інтерфейсів, вони відповідають його апаратним мережним пристроям.

На ***мережному рівні*** відбувається передавання пакетів із використанням різних транспортних технологій. Він забезпечує доставку даних між мережними інтерфейсами будь-яких хостів у неоднорідній мережі з довільною топологією. На цьому рівні реалізована адресація інтерфейсів і маршрутизація пакетів. Основним протоколом цього рівня у стеку TCP/IP є IP (Internet Protocol).

Транспортний рівень реалізує базові функції з організації зв'язку між ***процесами***, що виконуються на віддалених хостах. У стеку TCP/IP на цьому рівні функціонують протоколи ***TCP*** (Transmission Control Protocol) і ***UDP*** (User Datagram Protocol). TCP забезпечує надійне передавання повідомлень між віддаленими процесами користувача за рахунок утворення віртуальних з'єднань .

Прикладний рівень реалізує набір різноманітних мережних сервісів, наданих кінцевим користувачам і застосуванням. До цього рівня належать протоколи, реалізовані різними мережними застосуваннями (службами), наприклад, HTTP (основа організації Web), SMTP (основа організації пересилання електронної пошти).

2.2 Канальний рівень

Реалізація канального рівня включає драйвер мережного пристрою ОС і апаратний мережний пристрій та приховує від програмного забезпечення верхнього рівня і прикладних програм деталі взаємодії з фізичними каналами, надаючи їм абстракцію мережного інтерфейсу. Передавання даних мережею у програмному забезпеченні верхнього рівня відбувається між мережними інтерфейсами.

Кількість мережних інтерфейсів співвідноситься з кількістю мережних апаратних пристроїв хоста. Крім того, виділяють спеціальний ***інтерфейс зворотного зв'язку***; усі дані, передані цьому інтерфейсу, надходять на вхід реалізації стека протоколів того самого хоста.

2.3 Мережний рівень

Протокол IPv4

Протокол IP надає засоби доставки дейтаграм неоднорідною мережею без встановлення з'єднання. Він реалізує доставку за заданою адресою, але надійність, порядок доставки і відсутність дублікатів не гарантовані. Усі засоби щодо забезпечення цих характеристик реалізуються у протоколах вищого рівня (TCP).

Кожний мережний інтерфейс в IP-мережі має унікальну адресу. Такі адреси називають ***IP-адресами***. Стандартною версією є IP версії 4 (IPv4), де використовують адреси завдовжки 4 байти. Їх записують у крапково-десятковому поданні (чотири десяткові числа, розділені крапками, кожне з яких відображає один байт адреси). Прикладом може бути 194.41.233.1. Спеціальну адресу зворотного зв'язку 127.0.0.1 присвоюють інтерфейсу зворотного зв'язку і використовують для зв'язку із застосуваннями, запущеними на локальному хості.

Р доставляє дейтаграми мережному інтерфейсу. Пошук процесу на відповідному хості забезпечують протоколи транспортного рівня (TCP).

Протокол IPv6

Недоліком протоколу IPv4 є незначна довжина IP-адреси. Кількість адрес, які можна відобразити за допомогою 32 біт, є недостатньою з огляду на темпи росту Інтернету. Сьогодні нові IP-адреси виділяють обмежено.

Для вирішення проблеми запропоновано нову реалізацію IP-протоколу — *IP версії 6* (IPv6), відмінністю якої є довжина адреси — 128 біт (16 байт).

Інші протоколи мережного рівня

На мережному рівні реалізовано й інші протоколи. Для забезпечення мережної діагностики застосовують протокол ***ICMP*** (Internet Control Message Protocol), який використовують для передачі повідомлень про помилки під час пересилання IP-

дейтаграм, а також для реалізації найпростішого ***луна-протоколу,*** що реалізує обмін запитом до хосту і відповіддю на цей запит.

Сучасні ОС мають утиліту ***ping***, яку використовують для перевірки досяжності віддаленого хосту. Ця утиліта використовує луна-протокол у рамках ICMP.

2.4 Транспортний рівень

Протокол TCP

Пакет з TCP-заголовком називають ***TCP-сегментом***. Основні характеристики протоколу TCP такі.

Підтримка комунікаційних каналів між клієнтом і сервером, які називають ***з'єднаннями***. TCP-клієнт встановлює з'єднання з конкретним сервером, обмінюється даними з сервером через це з'єднання, після чого розриває його.

Забезпечення надійності передавання даних. Коли дані передають за допомогою TCP, потрібне підтвердження їхнього отримання. Якщо воно не отримане впродовж певного часу, пересилання даних автоматично повторюють, після чого протокол знову очікує підтвердження. Час очікування зростає зі збільшенням кількості спроб. Після певної кількості безуспішних спроб з'єднання розривають. Неповного передавання даних через з'єднання бути не може: або воно надійно пересилає дані, або його розривають.

Встановлення послідовності даних. Кожен сегмент, переданий за цим протоколом, супроводжує номер послідовності . Якщо сегменти приходять у невірному порядку, TCP на підставі цих номерів може переставити їх перед тим як передати повідомлення в застосування.

Керування потоком даних. Протокол TCP повідомляє віддаленому застосуванню, який обсяг даних можливо прийняти від нього у будь- який момент часу. Це значення називають ***оголошеним вікном***, воно дорівнює обсягу вільного простору у буфері, призначеному для отримання даних. Вікно динамічно змінюється: під час читання даних із буфера збільшується, у разі надходження даних мережею - зменшується. Це гарантує, що буфер не може переповнитися. Якщо буфер заповнений повністю, розмір вікна зменшують до нуля. Після цього TCP, пересилаючи дані, очікуватиме, поки у буфері не вивільниться місце.

TCP-з'єднання є ***повнодуплексними:*** з'єднання у будь-який момент часу можна використати для пересилання даних в обидва боки. TCP відстежує номери послідовностей і розміри вікон для кожного напрямку передавання даних.

Порти

Для встановлення зв'язку між двома процесами на транспортному рівні недостатньо наявності IP-адрес . Щоб розрізнити процеси, які виконуються на одному хості, використовують концепцію ***портів***.

Порти ідентифікують цілочисловими значеннями розміром 2 байти (від 0 до 65 535). Кожний порт унікально ідентифікує процес, запущений на хості: для того щоб TCP-сегмент був доставлений цьому процесові, у його заголовку зазначається цей порт. Процес-сервер використовує заздалегідь визначений порт, на який можуть вказувати клієнти для зв'язку із цим сервером. Для клієнтів порти зазвичай резервують динамічно.

Для деяких сервісів за замовчуванням зарезервовано конкретні номери портів у діапазоні від 0 до 1023; для протоколу HTTP (веб-серверів) це порт 80, а для протоколу SMTP — 25. Відомі порти розподіляються централізовано, подібно до IP-адрес. Якщо порт зайнятий деяким процесом, то інший процес на тому самому хості повторно зайняти його не зможе.

2.5 Передавання даних стеком протоколів Інтернету

Мережні протоколи стека TCP/IP використовуються для зв'язку між рівноправними сторонами, найчастіше такий зв'язок відбувається за принципом ***«клієнт-сервер»,*** коли одна сторона (***сервер***) очікує появи дейтаграм або встановлення з'єднання, а інша (***клієнт***) відсилає дейтаграми або створює з'єднання.

Основні етапи процесу обміну даними між клієнтом і сервером із використанням протоколу TCP/IP.

1.Застосування-клієнт (веб-браузер) у режимі користувача формує HTTP-запит до веб-сервера. Формат запиту визначений протоколом прикладного рівня (HTTP), зокрема у ньому зберігають шлях до потрібного документа на сервері. Після цього браузер виконує ряд системних викликів. При цьому у ядро ОС передають вміст HTTP-запиту, IP-адресу комп'ютера, на якому запущено веб-сервер, і номер порту, що відповідає цьому серверу.

2.Далі перетворення даних пакета відбувається в ядрі.

Спочатку повідомлення обробляють засобами підтримки протоколу транспортного рівня (TCP). Його доповнюють TCP-заголовком, що містить номер порту веб-сервера та інформацію, необхідну для надійного пересилання даних. HTTP-запит перетворюється у TCP-сегмент та стає корисним навантаженням — даними, які пересилають для обробки в режимі користувача.

TCP-сегмент обробляють засобами підтримки протоколу мережного рівня (IP). При цьому він перетворюється в IP-дейтаграму (його доповнюють IP-заголовком, що містить IP-адресу віддаленого комп'ютера та іншу інформацію, необхідну для передавання мережею).

IP-дейтаграма надходить на рівень драйвера мережного пристрою (Ethernet), який додає до неї інформацію, необхідну для передавання за допомогою Ethernet-пристрою. Пакет з Ethernet-інформацією називають ***Ethernet-фреймом***. Фрейм передають мережному пристрою, який відсилає його мережею. Фрейм містить адресу призначення у Ethernet, що є адресою мережної карти комп'ютера в тій самій локальній мережі (або іншого пристрою, який може переадресувати пакет далі в напрямку до місця призначення).

Апаратне забезпечення Ethernet забезпечує реалізацію передавання даних фізичною мережею у вигляді потоку бітів.

Дотепер пакет переходив від засобів підтримки протоколів вищого рівня до протоколів нижчого.

Тепер пакет переміщатиметься мережею. При цьому можуть здійснюватися різні його перетворення. Наприклад, коли Ethernet - фрейм доходить до адресата в мережі Ethernet, відповідне програмне або апаратне забезпечення виділяє IP-дейтаграму із фрейму, за IP-заголовком визначає, яким каналом відправляти її далі, перетворює дейтаграму відповідно до характеристик цього каналу (наприклад, знову в Ethernet-фрейм) і відсилає її в наступний пункт призначення. На шляху повідомлення може перейти в мережі, зв'язані модемами, і тоді формат зовнішньої оболонки буде змінено (наприклад, у формат протоколів SLIP або PPP), але вміст (IP-дейтаграма) залишиться тим самим.

Далі відбувається декілька етапів демультиплексування пакетів. Кажуть, що пакет піднімається у стеку протоколів:

Драйвер мережного пристрою Ethernet виділяє IP-дейтаграму із фрейму і передає її засобам підтримки протоколу IP.

Засоби підтримки IP перевіряють IP-адресу в заголовку, і, якщо вона збігається з локальною IP-адресою, виділяють TCP-сегмент із дейтаграми і передають його засобам підтримки TCP.

Засоби підтримки TCP визначають застосування-адресат за номером порту, заданим у TCP-заголовку (це веб-сервер, що очікує запитів від клієнтів). Після цього виділяють HTTP-запит із TCP-сегмента і передають його цьому застосуванню для обробки в режимі користувача.

Сервер обробляє HTTP-запит (наприклад, відшукує на локальному диску відповідний документ).

3 Система імен DNS

3.1 Система імен DNS

Доменна система імен (DNS) — це розподілена база даних, яку застосування використовують для організації відображення символьних імен хостів (доменних імен) на IP-адреси. За допомогою DNS завжди можна знайти IP-адресу, що відповідає заданому доменному імені. Розподіленість DNS полягає в тому, що немає жодного хосту в Інтернеті, який би мав усю інформацію про це відображення. Кожна група хостів підтримує свою власну базу даних імен, відкриту для запитів зовнішніх клієнтів та інших серверів. Підтримку бази даних імен здійснюють за допомогою застосування, яке називають ***DNS-сервером*** або ***сервером імен***.

Доступ до DNS з прикладної програми здійснюють за допомогою розпізнавача - клієнта, який звертається до DNS-серверів для перетворення доменних імен в IP-адреси (цей процес називають ***розв'язанням доменних імен)***. Розпізнавач реалізований як бібліотека, компонована із застосуваннями.

3.2 Простір імен DNS

Простір імен DNS є ієрархічним. Кожний вузол супроводжує символьна позначка. Коренем дерева є вузол із позначкою нульової довжини. Доменне ім'я будь-якого вузла дерева — це список позначок, починаючи із цього вузла (зліва направо) і до кореня, розділених символом «крапка». Наприклад, доменне ім'я «*staff.lpnu.ua*». Доменні імена мають бути унікальними.

Доменом називають піддерево ієрархічного простору імен.

Для позначення домену використовують доменне ім'я кореня цього піддерева: хост www.staff.lpnu.ua. належить домену із суфіксом *staff.lpnu.ua*, той, у свою чергу, - домену із суфіксом *lpnu.ua*. і т.д.

Доменне ім'я, що завершується крапкою, називають ***повним доменним, іменем***. Якщо крапка наприкінці імені відсутня, то це ім'я може бути доповнене (до нього може бути доданий суфікс відповідного домену). Такі імена можуть використовуватись у рамках домену. Серед доменів верхнього рівня виділяють усім відомі com, edu, org тощо, а також домени для країн (ua для України). Є спеціальний домен arpa, який використовують для зворотного перетворення IP-адрес у DNS-імена.

3.3 Розподіл відповідальності

Розподіл відповідальності за зони DNS-дерева -важлива характеристика доменної системи імен. Немає організації або компанії, яка б керувала відображенням для всіх позначок дерева. Є спеціальна організація (Network Information Center, NIC), що керує доменом верхнього рівня і делегує відповідальність іншим організаціям за інші зони. ***Зоною*** називають частину DNS-дерева, що адмініструється окремо. Прикладом зони є домен другого рівня (наприклад, lpnu.ua). Багато організацій розділяють свої зони на менші відповідно до доменів наступного рівня (наприклад, staff.lpnu.ua, vns.lpnu.ua тощо), аналогічним чином делегуючи відповідальність за них. У цьому разі зоною верхнього рівня вважають частину домена, що не включає виділені в ній зони.

Після делегування відповідальності за зону для неї необхідно встановити кілька серверів імен (як мінімум два — основний і резервний). Під час розміщення в мережі нового хоста інформація про нього повинна заноситься у базу даних основного сервера відповідної зони. Після цього інформацію автоматично синхронізують між основним і резервним серверами.

3.4 Отримання IP-адрес

Якщо сервер імен не має необхідної інформації, він її шукає на інших серверах. Процес отримання такої інформації називають ***ітеративним запитом***.

Приклад ітеративного запиту отримання IP-адреси для імені *www.staff.lpnu.ua*. Спочатку локальний сервер зв'язується із кореневим сервером імен, відповідальним за домен верхнього рівня. Кореневий сервер імен зберігає інформацію про сервери першого рівня. Отримавши запит на відображення імені, він визначає, що це ім'я належить не до його зони відповідальності, а до домену .ua, і повертає локальному серверу інформацію про адреси та імена всіх серверів відповідної зони. Далі локальний сервер звертається до одного із цих серверів з аналогічним запитом. Той сервер містить інформацію про те, що для зони *lpnu.ua* є свій сервер імен, у результаті локальний сервер отримує адресу цього сервера. Процес повторюють доти, поки запит не надійде на сервер, відповідальний за домен *staff.lpnu.ua*, що може повернути коректну IP-адресу.

3.5 Кешування IP-адрес

Кешування IP-адрес дозволяє значно зменшити навантаження на мережу. Коли сервер імен отримує інформацію про відображення (наприклад, IP-адресу, що відповідає доменному імені), він зберігає цю інформацію локально (у спеціальному кеші). Наступний аналогічний запит отримає у відповідь дані з кеша без звертання до інших серверів. Інформацію зберігають у кеші обмежений час. Коли для сервера не задана зона, за яку він відповідає, кешування є його єдиним завданням. Це поширений ***сервер кешування***.

Типи DNS-ресурсів

лемент інформації у базі даних DNS називають ***ресурсним записом***. Кожний такий запис має клас і тип. Для записів про відображення IP-адрес класом завжди є IN.

Розглянемо типи DNS-ресурсів. Найважливішим є ***A-запис***, що пов'язує повне доменне ім'я з IP-адресою. Саме на основі таких записів сервери імен повертають інформацію про відображення. Ще одним типом запису є ***CNAME-запис***, що задає ***аліас доменного імені***. Одній і тій самий IP-адресі (A-запису) може відповідати кілька таких аліасів.

4 Програмний інтерфейс сокетів Берклі

Програмний інтерфейс сокетів Берклі —засіб зв'язку між прикладним рівнем мережної архітектури TCP/IP і транспортним рівнем. Причина такого розташування полягає в тому, що це засіб зв'язку між кодом застосування (який виконують на прикладному рівні) і реалізацією стека протоколу ОС (найвищим рівнем якої є транспортний). Цей інтерфейс є набором системних викликів ОС.

Назва цього API пов'язана з тим, що він був розробленій Каліфорнійським університетом у Берклі.

4.1 Особливості роботи з адресами

Відображення адрес сокетів

Більшість системних викликів роботи із сокетами приймають як параметри покажчики на спеціальні структури даних, що відображають адреси сокетів. У разі використання TCP/IP зі звичайним IP-протоколом (IPv4) адресу задають структурою sockaddr_in. Вона визначена у заголовному файлі <netinet/in.h> і містить такі поля:

```
typedef struct sockaddr_in {
  short          sin_family;
  u_short        sin_port;
  struct in_addr sin_addr;
  char           sin_zero[8];
} SOCKADDR_IN, *PSOCKADDR_IN, *LPSOCKADDR_IN;
```

sin_family — для TCP/IP має дорівнювати константі AF_INET;

sin_port - цілочисловий номер порту для TCP або UDP;

sin_addr - відображення IP-адреси (структура *in_addr* з одним полем *s_addr*).

Робота з IP-адресами

У протоколі IPv4 IP-адреса є цілим значенням розміром 4 байти. Для того щоб перетворити крапково-десяткове подання такої адреси (n. n. n. n) у ціле число, слід використати функцію inet_aton():

#include <arpa/inet.h>

*int inet_aton(const char *c_ip, struct in_addr *s_ip);*

c_ip - рядкове відображення IP-адреси: "192.168.10.28";

s_ip — структура *in_addr*, що міститиме цілочислове відображення адреси, розташоване в пам'яті відповідно до мережного порядку байтів (поле *s_addr*);

цю структуру заповнюють усередині виклику.

У разі успішного перетворення ця функція повертає ненульове значення, у разі помилки — нуль.

4.2 Створення сокетів

Перший етап - створити дескриптор сокета за допомогою системного виклику socket ():

#include <sys/socket.h>

int socket(int family, int type, int protocol);

family — комунікаційний домен (AF_INET — домен Інтернету на основі IPv4);

type - тип сокета (SOCK_STREAM - потоковий сокет, S0CK_DGRAM - дейтаграмний сокет, S0CK_RAW — *сокет прямого доступу* , призначений для роботи із протоколом мережного рівня, наприклад IP);

protocol — тип протоколу (0 - протокол вибирають автоматично, виходячи з домену і типу сокета). Приклад використання socket():

int sockfd = socket(AF_INET, SOCK_STREAM, 0);

Виклик *socket()* повертає цілочисловий дескриптор сокета або -1, якщо сталася помилка. Цей сокет можна використовувати для різних цілей: організації очікування з'єднань сервером, задання з'єднання клієнтом, отримання інформації про мережну конфігурацію хоста.

Для системних викликів інтерфейсу сокетів Берклі використовується заголовний файл <*sys/socket. h*>.

4.3 Робота з потоковими сокетами

Зв'язування сокета з адресою

Для пов'язування сокета з адресою необхідно використати системний виклик bind():

```
int bind(
  [in] SOCKET         s,
       const sockaddr *addr,
  [in] int            namelen
);
```

*int bind(int sockfd. const struct sockaddr *pmy_addr, socklen_t alen);*

sockfd — дескриптор сокета, створений за допомогою socket();

pmy_addr - покажчик на структуру, що задає адресу сокета (наприклад, sockaddr_in);

alen — розмір структури, на яку вказує pmy_addr.

Цей виклик повертає -1, якщо виникла помилка.

Під час виконання bind() застосуванням-сервером необхідно задати наперед відомий порт, через який клієнти зв'язуватимуться з ним. Можна також вказати конкретну IP-адресу, найчастіше достатньо взяти довільну адресу локального хоста, скориставшись константою INADDR_ANY:

struct sockaddr_in my_addr = { 0 };

int listenfd = socket(...);

// ... задача my_addr.sin_family і my_addr.sin_port my_addr.sin_addr.s_addr = INADDR_ANY;

*bind(listenfd, (struct sockaddr *)&my_addr, sizeof(my_addr));*

Найпоширенішою помилкою під час виклику bind() є помилка з кодом EADDRINUSE, яка свідчить про те, що цю комбінацію «IP-адреса — номер порту» вже використовує інший процес. Часто це означає повторну спробу запуску того самого сервера.

if (bind(listenfd, ...)== -1&& errno == EADDRINUSE)

{ printf(“помилка, адресу вже використовують"); exit(-l); }

Щоб досягти гнучкості у вирішенні цієї проблеми, рекомендують дозволяти користувачам налаштовувати номер порту (задавати його у конфігураційному файлі, командному рядку тощо).

Відкриття сокета для прослуховування

Сокет, створений викликом *socket()* є клієнтським активним сокетом, за допомогою якого передбачають з'єднання із сервером. Щоб перетворити такий сокет у серверний прослуховувальний, призначений для прийому запитів на з'єднання від клієнтів, необхідно використати системний виклик listen():

```
int WSAAPI listen(
  [in] SOCKET s,
  [in] int    backlog
);
```

int listen(int sockfd, int backlog);

sockfd - дескриптор сокета, пов'язаний з адресою за допомогою bind();

*backl*og — задає довжину черги запитів на з'єднання для цього сокета. Внаслідок виклику *listen()* створяться дві черги запитів: перша з них містить запити, для яких процес з'єднання ще не завершений, друга — запити, для яких з'єднання встановлене. Параметр *backlog* визначає сумарну довжину цих двох черг; для серверів значного навантаження, рекомендують задавати велике значення аргументу:

listen(listenfd, 1024);

Внаслідок виклику *listen()* сервер готовий до прийому запитів на з'єднання.

Прийняття з'єднань

Після створення нове з'єднання потрапляє у чергу встановлених з'єднань. Для сервера воно залишається недоступним. Щоб отримати доступ до з'єднання, на сервері його потрібно прийняти за допомогою системного виклику *accept():*

```
SOCKET WSAAPI accept(
  [in]      SOCKET   s,
  [out]     sockaddr *addr,
  [in, out] int      *addrlen
);
```

*int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);*

sockfd — дескриптор прослуховувального сокета;

cliaddr — покажчик на структуру, у яку буде занесена адреса клієнта, що запросив з'єднання;

addrlen — покажчик на змінну, котра містить розмір структури *cliaddr*. Головною особливістю цього виклику є повернене значення — дескриптор нового сокета з'єднання, який створений внаслідок виконання цього виклику і призначений для обміну даними із клієнтом. Прослуховувальний сокет після виконання *accept ( )* готовий приймати запити на нові з'єднання.

```
struct sockaddr_in their_addr = { 0 };
int listenfd, connfd, sin_size = sizeof(struct sockaddr_in);
// listenfd = socket(...), bind(listenfd, ...), listen(listenfd, ...)
connfd = accept(listenfd, (struct sockaddr *)&their_addr, &sin_size);
// використання connfd для обміну даними із клієнтом
```

Виклик *accept()* приймає з'єднання, що стоїть першим у черзі встановлених з'єднань. Якщо вона порожня, процес переходить у стан очікування, де перебуватиме до появи нового запиту. Саме на цьому системному виклику очікують ітеративні та багатопотокові сервери, що використовують сокети.

Задання з'єднань на клієнті

Після створення сокета за допомогою socket() необхідно встановити для нього з'єднання за допомогою системного виклику conecte(), після чого можна відразу обмінюватися даними між клієнтом і сервером.

*int connect(int sockfd, const struct sockaddr *saddr. socklen_t alen);*

```
int WSAAPI connect(
  [in] SOCKET         s,
  [in] const sockaddr *name,
  [in] int            namelen
);
```

sockfd — сокет, створений за допомогою socket();

saddr — покажчик на структуру, що задає адресу сервера (IP-адресу віддаленого хоста, на якому запущено сервер і порт, який прослуховує сервер);

alen — розмір структури saddr. Приклад створення з'єднання на клієнті:

```
struct sockaddr_in their_addr = { 0 };
 sockfd = sockete(...);
```

```
// заповнення their_addr,sin_family, their_addr.sin_port
inet_aton(" IP-адреса-сервера", &(their_addr.sin_addr));
 connect(sockfd, (struct sockaddr *)&their_addr, sizeof(their_addr));
// обмін даними через sockfd
```

У разі виклику connecte() починають створювати з'єднання. Повернення відбувається, якщо з'єднання встановлене або сталася помилка (поверненим значенням буде -1). Якщо connecte () повернув помилку, користуватися цим сокетом далі не можна, його необхідно закрити. Коли потрібно повторити спробу, створюють новий сокет за допомогою socket().

Клієнтові немає потреби викликати bind() перед викликом connecte() — ядро системи автоматично виділяє тимчасовий порт для клієнтського сокета.

Закриття з'єднань

Після використання всі з'єднання необхідно закривати. Для цього застосовують стандартний системний виклик *close( ):*

```
int closesocket(
  [in] SOCKET s
);
```

Обмін даними між клієнтом і сервером

Після того як з'єднання було встановлене і відповідний сокет став доступний серверу (клієнт викликав connect(), а сервер – accept()), можна обмінюватися даними між клієнтом і сервером. Для цього використовують стандартні системні виклики read() і write(), а також спеціалізовані виклики recv() і send():

//прийняти nbytes або менше байтів із sockfd і зберегти їх у buf

*ssize_t recv(int sockfd, void *buf, size_t bytes_read, int flags);*

// відіслати nbytes або менше байтів із buf через sockfd

*ssize_t send(int sockfd, const void *buf, size_t bytes_sent,int flags);*

Виклики recv() і send() відрізняються від read() і write() параметром flags, що задає додаткові характеристики передавання даних. Тут задаватимемо цей параметр, що дорівнює нулю:

int bytes_received = recv(sockfd, buf, sizeof(buf), 0);

int bytes_sent = send(sockfd, buf, sizeof(buf), 0);

5 Архітектура мережної підтримки Windows XP

Компоненти мережної підтримки

Мережні API надають засоби для доступу до мережі із прикладних програм, незалежні від протоколів. Такі API частково реалізовані в режимі користувача. Серед мережних API використовують інтерфейси поіменованих каналів, Windows Sockets та віддаленого виклику процедур (RPC).

Драйвери транспортних протоколів приймають ARP-пакети від драйверів мережних API і обробляють запити, що містяться там, як вимагають реалізовані в них протоколи (TCP, IP тощо). Під час цієї обробки може знадобитися обмін даними через мережу, додавання або вилучення заголовків пакетів, взаємодія із драйверами мережних апаратних пристроїв. Драйвери протоколів відповідають за підтримку мережної взаємодії організацією повторного збирання пакетів, встановлення їхньої послідовності, відсилання повторних пакетів і підтверджень. Усі драйвери транспортних протоколів надають драйверам мережних API універсальний *інтерфейс транспортного драйвера*. Прикладом драйвера транспортного протоколу є драйвер підтримки TCP/IP (tcpip.sys).

Мініпорт-драйвери NDIS відповідають за організацію взаємодії драйверів транспортних протоколів і мережного апаратного забезпечення. Вони взаємодіють з іншими компонентами режиму ядра винятково за допомогою функцій спеціальної бібліотеки NDIS (ndis.sys), що реалізують універсальну *специфікацію інтерфейсу мережного драйвера* (Network Driver Interface Specification, NDIS). Деякі з цих функцій призначені для доступу із драйверів протоколів до засобів мініпорт-драйверів, інші мініпорт-драйвери викликають для безпосереднього доступу до мережних апаратних пристроїв через HAL. Прикладами мініпорт-драйверів NDIS є драйвери конкретних мережних адаптерів.

Спільне використання файлів і принтерів

Основою підтримки спільного використання файлів і принтерів є спеціальний протокол ***спільної файлової системи Інтернету*** (Common Internet File System, CIFS). Він визначає правила взаємодії з файловими клієнтами і серверами. Реалізація CIFS для Windows складається із двох частин — клієнтської (редиректор) і серверної (сервер), реалізованих як драйвери файлових систем.

Драйвер редиректора перехоплює запити на виконання файлового введення-виведення (наприклад, виклик функцій ReadFile(), WriteFile() або звертання до драйвера принтера), перетворює їх у CIFS-повідомлення і пересилає на віддалений хост, використовуючи драйвер транспортного протоколу. Драйвер сервера приймає

повідомлення від драйвера протоколу, перетворює їх назад у запити введення-виведення і передає драйверу локальної файлової системи (наприклад, NTFS) або драйверу принтера. Повернення даних відбувається у зворотному порядку.

6 Програмний інтерфейс Windows Sockets

6.1 Архітектура підтримки Windows Sockets

У режимі користувача Winsock API підтримують кількома системними DLL. Найважливіша з них — ws2_32.dll, де реалізовано основні функції, використовувані у застосуваннях. На рівні ядра підтримку Windows Sockets здійснюють драйвером файлової системи afd.sys, що реалізує операції із сокетами і звертається для пересилання даних до відповідного драйвера транспортного протоколу (драйвер підтримки TCP/IP).

Інтерфейс Winsock API посідає таке саме місце у багаторівневій мережній архітектурі, що й інтерфейс сокетів Берклі. Він розташований між рівнем застосувань і транспортним рівнем. У ньому відсутні засоби безпосередньої підтримки протоколів прикладного рівня, вона може бути реалізована на основі інтерфейсу Winsock.

6.2 Основні моделі використання Windows Sockets

Реалізація синхронного введення-виведення ґрунтується на сокетах із блокуванням, інтерфейс яких не відрізняється від інтерфейсу сокетів Берклі. їхнє використання можливе, коли не потрібні висока продуктивність і масштабованість застосування. На основі сокетів із блокуванням можна реалізовувати багатопотокові сервери. Підтримують і реалізацію серверів із повідомленням на основі select().

Ефективною і складнішою технологією, яка також реалізована у Windows Sockets, є *асинхронні сокети.* Є два підходи до їхнього використання. Перший із них використовує повідомлення на основі повідомлень і ґрунтується на функції ***WSAAsynchSelect().*** *І*нформацію про події, що відбулися із сокетом, передають у вигляді повідомлень Windows у віконну процедуру, де вона може бути оброблена. За другого підходу - повідомлення на основі подій (що використовує функцію *WSAEventSelect ())* - аналогічну інформацію передають сигналізацією об'єктів подій. Цей підхід з'явився у Winsock 2.

Найефективнішим підходом, реалізованим у Winsock 2, є сокети із перекриттям, що застосовують асинхронне введення-виведення. Сокети також можна використовувати у поєднанні з портами завершення введення-виведення.

6.3 Сокети з блокуванням

Інтерфейс сокетів із блокуванням майже збігається з інтерфейсом Berkeley Sockets. Проте є деякі відмінності:

Використання сокетів Берклі вимагає підключення набору різних заголовних файлів. Windows Sockets звичайно потребує лише заголовних файлів <winsock.h> або <winsock2.h>. Під час компонування потрібно задавати бібліотеку підтримки Winsock (ws2_32.lib для версії 2). Перед використанням будь-яких засобів Windows Sockets необхідно ініціалізувати бібліотеку, після використання - вивільнити зайняті ресурси. Для цього є функції WSAStartup() і WSACleanup():

```
#include <winsock.h>
WSADATA wsadata;
WSAStartup(HAKEWORDCl.l), Swsadata);
// ... робота із засобами Winsock
WSACleanup();
```

6.4 Асинхронні сокети

Для кожного сокета, стан якого необхідно відстежувати, необхідно створити об'єкт-подію, що належить до типу WSAEVENT.

```
WSAEVENT event = WSACreateEvent() ;
```

Після створення події необхідно пов'язати її із сокетом, тобто вказати, які зміни стану цього сокета призводитимуть до сигналізації події. Для встановлення такого зв'язку використовують функцію *WSAEventSelect():*

```
int WSAEventSelect(SOCKET sock, WSAEVENT event, long net_events);
```

net_events — бітова маска мережних подій. Кожній події відповідає певний біт цієї маски. Застосування очікуватиме отримання повідомлень про події, біти яких задані в масці.

Серед прапорів, що визначають біти для подій, можна виділити:

FD_READ — готовність до читання даних із сокета;
FD_WRITE — готовність до записування даних у сокет;
FD_ACCEPT — наявність нового з'єднання для сокета;
FD_CL0SE — закриття з'єднання.

Приклади розробки програм

Приклад 1 .Програма TCP – сервера

```
//Програма TCP_сервер
#include <winsock2.h>
#include <stdio.h>
#include <windows.h>

//прототипи функцій
//функція форматованого виводу кирилиці на консоль
int myprintf (const char *lpFormat, ... );

//функція  розшифровує помилки
char* encodeWSAGetLastError(int n);

void main(void)
{
  WSADATA wsaData;
  SOCKET ListeningSocket;
  SOCKET NewConnection;
  struct sockaddr_in ServerAddr;
  struct sockaddr_in ClientAddr;
  int ClientAddrLen;
  int Port = 5150;
  int Ret;
  char DataBuffer[1024];
  ////////ZeroMemory(ServerAddr, sizeof(ServerAddr));
  ////////ZeroMemory(ClientAddr, sizeof(ClientAddr));

  // Ініціалізація Winsock версії 2.2
  if ((Ret = WSAStartup(MAKEWORD(2,2), &wsaData)) != 0)
  {
        //Так як Winsock не завантажився, ми не можемо
    //використовувати WSAGetLastError        //для визначення
    //помилки. Код помилки повертає сама функция WSAStartup
```

```
        myprintf("WSAStartup помилка %s\n",
                    encodeWSAGetLastError(Ret));
        return;
}

// Створимо новий TCP сокет для  для прийому запитів на
//з'єднання від клієнтів.
if ((ListeningSocket = socket(AF_INET, SOCK_STREAM,
                    IPPROTO_TCP)) == INVALID_SOCKET)
{
        myprintf("Помилка socket %d\n",
            encodeWSAGetLastError(WSAGetLastError()));
        WSACleanup();
        return;
}

// Заповнюємо struct sockaddr_in ServerAddr, яка скаже функції
//bind, що ми хочемо слухати з'єднання на всіх інтерфейсах
//(INADDR_ANY), використовуючи порт 5150.
// Ми перетворюємо порядок байтів з системного у мережний
//(htons та htonl)
ServerAddr.sin_family = AF_INET;
ServerAddr.sin_port = htons(Port);
ServerAddr.sin_addr.s_addr = htonl(INADDR_ANY);

// Функція bind прив'язує адресну інформацію, визначену в
//ServerAddr до сокету ListeningSocket
if (bind(ListeningSocket, (struct sockaddr *)&ServerAddr,
                sizeof(ServerAddr)) == SOCKET_ERROR)
{
        myprintf("Помилка bind %s\n",
              encodeWSAGetLastError(WSAGetLastError()));
```

```
        closesocket(ListeningSocket);
        WSACleanup();
        return;
}

// Робимо сокет пасивним для прослуховування (прийому) запитів
//на TCP-з'єднання від клієнтів. Довжина черги запитів на
//з'єднання дорівнює 5
if (listen(ListeningSocket, 5) == SOCKET_ERROR)
{
        myprintf("Помилка listen %s\n",
             encodeWSAGetLastError(WSAGetLastError()));
        closesocket(ListeningSocket);
        WSACleanup();
        return;
}
myprintf("Чекаємо з'єднання на порту %d.\n", Port);
ClientAddrLen = sizeof(ClientAddr);

// Приймаємо нове з'єднання, коли воно виникне
if ((NewConnection =accept(ListeningSocket,(struct sockaddr *)
          &ClientAddr,&ClientAddrLen)) == INVALID_SOCKET)
{
        myprintf("Помилка accept %s\n",
             encodeWSAGetLastError(WSAGetLastError()));
        closesocket(ListeningSocket);
        WSACleanup();
        return;
}
myprintf("Успішно з'єдналися з %s:%d.\n",
inet_ntoa(ClientAddr.sin_addr), ntohs(ClientAddr.sin_port));
```

```
// Далі ми можемо знову очікувати на сокеті ListeningSocket
//нові з'єднання знову, викликаючи accept та/або почати
//передавати та приймати дані на сокеті NewConnection.
// Для простоти зупинимо прослуховування, закриваючи сокет
//ListeningSocket. Можна починати прийом та передачу даних на
//сокеті NewConnection.

closesocket(ListeningSocket);

// Для простоти обмежимося лише прийомом
myprintf("Ждемо дані для отримання..\n");
if ((Ret = recv(NewConnection, DataBuffer, sizeof(DataBuffer),
                                0))== SOCKET_ERROR)
{
        myprintf("Помилка recv %s\n",
                encodeWSAGetLastError(WSAGetLastError()));
        closesocket(NewConnection);
        WSACleanup();
        return;
}

// Створюємо з отриманих даних рядок мови C
DataBuffer[Ret] = '\0';
myprintf("Успішно отримано %d байтів в повідомленні %s.\n",
                                Ret, DataBuffer);
myprintf("Закрываем соединение с клиентом.\n");
closesocket(NewConnection);

// Вивантажуємо Winsock
WSACleanup();
myprintf("Натисніть Enter для завершення.\n");
getchar();
```

```
}



//опис функції форматованого виводу кирилиці на консоль,
int myprintf (const char *lpFormat, ... )
{
  int nLen = 0;
  int nRet = 0;
  char ansiBuffer[512] ;
  char oemBuffer[512] ;
  va_list arglist ;
  HANDLE hOut = NULL;
  ZeroMemory(ansiBuffer, sizeof(ansiBuffer));
  va_start(arglist, lpFormat);
  nLen = lstrlen( lpFormat ) ;
  nRet = wvsprintf( ansiBuffer, lpFormat, arglist );
  if( nRet >= nLen || GetLastError() == 0 )
  {
        hOut = GetStdHandle(STD_OUTPUT_HANDLE) ;
        CharToOem(ansiBuffer,oemBuffer);
        if( hOut != INVALID_HANDLE_VALUE )
              WriteConsole( hOut, oemBuffer, lstrlen(oemBuffer), (LPDWORD)&nLen, NULL ) ;
  }
  return nLen ;
}
//опис функції  розшифровування помилок
char* encodeWSAGetLastError(int n)
{
  switch(n)
  {
        case WSAEINTR: return "WSAEINTR: Перерваний виклик функції."; break;
        case WSAEACCES: return "WSAEACCES: Доступ заборонений."; break;
```

```
        default : return "Невідома помилка"; break;
  }
}
```

Приклад 2 .Програма TCP – клієнта

```
//Програма TCP_клієнт
#include <winsock2.h>
#include <stdio.h>
#include <windows.h>

//прототипи функцій
//функція форматованого виводу кирилиці на консоль,
int myprintf (const char *lpFormat, ... );

//функція  розшифровує помилки
char* encodeWSAGetLastError(int n);

void main(int argc, char **argv)
{
  WSADATA wsaData;
  SOCKET s;
  struct sockaddr_in ServerAddr, ca;
  int Port = 5150;
  int Ret;
  char *msg = "Привіт";
  int lmsg = strlen(msg);
  int caSize = sizeof(ca);
  ///ZeroMemory(ServerAddr, sizeof(ServerAddr));
  /////ZeroMemory(ca, sizeof(caSize));
  if (argc <= 1)
  {
        myprintf("Використання: tcpclient <Server IP address>.\n");
```

```
        return;
}

// Ініціалізація Winsock версии 2.2
if ((Ret = WSAStartup(MAKEWORD(2,2), &wsaData)) != 0)
{
        //Так як Winsock не завантажився, ми не можемо
   //використовувати WSAGetLastError для визначення помилки.
   //Код помилки повертає сама функция WSAStartup

        myprintf("WSAStartup ошибка %d\n",
                        encodeWSAGetLastError(Ret));
        return;


   //Створимо новий сокет для з'єднання цього TCP клієнта
        if ((s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) ==
                                INVALID_SOCKET)
        {
                myprintf("socket ошибка %d\n",
            encodeWSAGetLastError(WSAGetLastError()));
                WSACleanup();
                return;
        }
       }

       //Заповнимо у змінній ServerAddr типу struct sockaddr_in
 //адресну інформацію про сервер, до якого ми хочемо
 //підключитися. Сервер слухає порт 5150. IP-адреса сервера
 //задається в командной строке

       ServerAddr.sin_family = AF_INET;
```

```
        ServerAddr.sin_port = htons(Port);
        ServerAddr.sin_addr.s_addr = inet_addr(argv[1]);

 //З'єднаємо сокет s з сервером
 myprintf("Спроба з'єднання до %s:%d...\n",
 inet_ntoa(ServerAddr.sin_addr), htons(ServerAddr.sin_port));

 if (connect(s, (struct sockaddr*) &ServerAddr,
                    sizeof(ServerAddr)) == SOCKET_ERROR)
{
        myprintf("Помилка connect %s\n",
            encodeWSAGetLastError(WSAGetLastError()));
        closesocket(s);
        WSACleanup();
        return;
}
myprintf("Наше з'єднання успішне.\n");

// Інформацію про истинну адресу та порт локального
//з'єднананого сокета sc можна отримати  так

getsockname(s, (struct sockaddr *)&ca, &caSize);
myprintf("Локальный адрес:%s и порт:%d сокета
     клиента\n",inet_ntoa(ca.sin_addr),ntohs(ca.sin_port));

// Тепер ми можемо обмінюватися даними з сервером через сокет s
// Ми просто передамо серверу повідомлення "Привіт"
myprintf("Спроба передати повідомлення Привіт.\n");
if ((Ret = send(s, msg, lmsg, 0)) == SOCKET_ERROR)
{
        myprintf("Ошибка send %s\n",
             encodeWSAGetLastError(WSAGetLastError()));
```

```
            closesocket(s);
            WSACleanup();
            return;
    }
    myprintf("Успішно передано %d байтів повідомлення %s.\n", Ret,
                                        msg);
    myprintf("Закриваємо з'єднання з сервером.\n");
    closesocket(s);

    // Вивантажуємо Winsock
    WSACleanup();
    myprintf("Натисніть Enter для завершення.\n");
    getchar();
}

//опис функції форматованого виводу кирилиці на консоль,
int myprintf (const char *lpFormat, ... )
{
    int nLen = 0;
    int nRet = 0;
    char ansiBuffer[512] ;
    char oemBuffer[512] ;
    va_list arglist ;
    HANDLE hOut = NULL;
    ZeroMemory(ansiBuffer, sizeof(ansiBuffer));
    va_start(arglist, lpFormat);
    nLen = lstrlen( lpFormat ) ;
    nRet = wvsprintf( ansiBuffer, lpFormat, arglist );
    if( nRet >= nLen || GetLastError() == 0 )
    {
            hOut = GetStdHandle(STD_OUTPUT_HANDLE) ;
            CharToOem(ansiBuffer,oemBuffer);
```

```
        if( hOut != INVALID_HANDLE_VALUE )
                WriteConsole( hOut, oemBuffer, lstrlen(oemBuffer),
                        (LPDWORD)&nLen, NULL ) ;
  }
  return nLen ;
}

//опис функції  розшифровування помилок
char* encodeWSAGetLastError(int n)
{
  switch(n)
  {
        case WSAEINTR: return "WSAEINTR: Перерваний виклик
                        функції."; break;
        case WSAEACCES: return "WSAEACCES: Доступ заборонений.";
                                break;
        default : return "Невідома помилка"; break;
  }
}
```

Приклад 3 . Програма, що виводить всі IP - адреси сайту Google

```
#include <winsock2.h>
#include <stdio.h>
#include <conio.h>
main()
{
  char **pptr;
  struct hostent *hptr;
  struct in_addr sin_addr;
  WSADATA wsd;
  WSAStartup(MAKEWORD(2,2), &wsd);
```

```
  hptr = gethostbyname("google.com");
  pptr = hptr->h_addr_list; // Список IP-адрес хоста google.com

  for ( ; *pptr != NULL; pptr++) // перебираємо всі IP-адреси
  {
        CopyMemory(&sin_addr, *pptr, hptr->h_length);
    //Копіюємо IP-адресу у структуру sin_addr
        printf("\taddress: %s\n",inet_ntoa(sin_addr));
    //Перетворюємо адресу в точечну нотацію
  }
  getch();
}
```

Завдання для студентів

1. Розробити програми роботи керування мережними засобами ОС. Для програмної реалізації використовувати середовище програмування Microsoft Visual Studio, мова програмування – C/C++, інтерфейс консольний.
2. Скласти звіт про виконання лабораторної роботи. Зміст звіту:

➢ опис функцій для роботи з мережними засобами;

➢ постановка задачі - класичним прикладом використання механізму сокетів є побудова системи клієнт-серверної архітектури, за якою один додаток грає роль сервера, що виконує запити другого додатка;

➢ програмний код;

➢ результати виконання програми;

➢ висновок.

3. До захисту лабораторної роботи підготувати відповіді на контрольні питання.

Контрольні питання

1. Охарактеризувати загальні принципи мережної підтримки.
2. Назвати рівні мережної архітектури і мережні сервіси.
3. Визначити та охарактеризувати мережні протоколи.
4. Як виконується реалізація стека протоколів Інтернету?
5. Назвати рівні мережної архітектури TCP/IP.
6. Охарактеризувати канальний рівень.
7. Охарактеризувати мережний рівень.
8. Охарактеризувати транспортний рівень.

9. Як виконується передавання даних стеком протоколів Інтернету?
10. Навести визначення та характеристику системи імен DNS .
11. Охарактеризувати простір імен DNS.
12. Як виконується розподіл відповідальності?
13. Як виконується кешування IP-адрес?
14. Навести визначення та характеристику програмного інтерфейсу сокетів Берклі.
15. Назвати особливості роботи з адресами.
16. Як виконується створення сокетів?
17. Як виконується Робота з потоковими сокетами?
18. Навести визначення та характеристику програмного інтерфейсу Windows Sockets.
19. Охарактеризувати архітектуру підтримки Windows Sockets.
20. Охарактеризувати основні моделі використання Windows Sockets .
21. Охарактеризувати сокети з блокуванням.
22. Охарактеризувати асинхронні сокети.

Варіанти завдань:

Розробити дві програми, яка демонструють роботу використання механізму сокетів. Одна програма виконує роль сервера, інша – роль клієнта. Індивідуальні завдання такі ж, як до лабораторної роботи №1. Клієнт запитує у користувача вхідні дані і передає їх серверу. Сервер виконує основні обчислення і результат передає клієнту. Клієнт виводить результат на екран.

Приклад виконання:

```
// TCP сервер
#pragma comment(lib, "Ws2_32.lib")

#define _WINSOCK_DEPRECATED_NO_WARNINGS

#include <winsock2.h>
#include <stdio.h>
#include <windows.h>


int main(void)
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	printf("TCP Сервер \n");
	printf("___________\n\n");

	WSADATA wsaData;
	SOCKET ListeningSocket;
	SOCKET NewConnection;
	struct sockaddr_in ServerAddr;
	struct sockaddr_in ClientAddr;
	int ClientAddrLen;
	u_short Port = 5150;
	int Ret;
	char DataBuffer[1024];
	char* msg = (char*)"Привіт, я TCP сервер!";

	// Ініціалізація Winsock версії 2.2
	if ((Ret = WSAStartup(MAKEWORD(2, 2), &wsaData)) != 0)
	{
		//Winsock не завантажився. Код помилки повертає функция WSAStartup
		printf("Помилка WSAStartup, номер помилки %d\n", Ret);
		return -1;
	}

	// Створимо новий TCP сокет для для прийому запитів на з'єднання від клієнтів.
	if ((ListeningSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) ==
INVALID_SOCKET)
	{
		printf("Помилка socket, номер помилки %d\n", WSAGetLastError());
		WSACleanup();
		return -2;
	}

	// Заповнюємо struct sockaddr_in ServerAddr, яка скаже функції
	//bind, що ми хочемо слухати з'єднання на всіх інтерфейсах
	//(INADDR_ANY), використовуючи порт 5150.
	// Ми перетворюємо порядок байтів з системного у мережний (htons та htonl)
	ServerAddr.sin_family = AF_INET;
	ServerAddr.sin_port = htons(Port);
	ServerAddr.sin_addr.s_addr = htonl(INADDR_ANY);

	// Функція bind прив'язує адресну інформацію, визначену в ServerAddr до сокету
ListeningSocket
	if (bind(ListeningSocket, (struct sockaddr*)&ServerAddr, sizeof(ServerAddr)) ==
SOCKET_ERROR)
	{
		printf("Помилка bind, номер помилки %d\n", WSAGetLastError());
		closesocket(ListeningSocket);
		WSACleanup();
```

```
		return -3;
	}
	// Робимо сокет пасивним для прослуховування (прийому) запитів на TCP-з'єднання від клієнтів. Довжина черги запитів на
	//з'єднання дорівнює 5
	if (listen(ListeningSocket, 5) == SOCKET_ERROR)
	{
		printf("Помилка listen, номер помилки %d\n", WSAGetLastError());
		closesocket(ListeningSocket);
		WSACleanup();
		return -4;
	}
	printf("Чекаємо з'єднання на порту %d.\n", Port);
	ClientAddrLen = sizeof(ClientAddr);
	// Приймаємо нове з'єднання, коли воно виникне
	if ((NewConnection = accept(ListeningSocket, (struct sockaddr*)&ClientAddr, &ClientAddrLen)) == INVALID_SOCKET)
	{
		printf("Помилка accept, номер помилки %d\n", WSAGetLastError());
		closesocket(ListeningSocket);
		WSACleanup();
		return -5;
	}
	printf("Успішно з'єдналися з %s:%d.\n", inet_ntoa(ClientAddr.sin_addr), ntohs(ClientAddr.sin_port));

	// Далі ми можемо знову очікувати на сокеті ListeningSocket
	//нові з'єднання знову, викликаючи accept та/або почати
	//передавати та приймати дані на сокеті NewConnection.
	// Для простоти зупинимо прослуховування, закриваючи сокет
	//ListeningSocket. Можна починати прийом та передачу даних на
	//сокеті NewConnection.
	closesocket(ListeningSocket);

	printf("Чекаємо дані для отримання...\n");
	if ((Ret = recv(NewConnection, DataBuffer, sizeof(DataBuffer), 0)) == SOCKET_ERROR)
	{
		printf("Помилка recv, номер помилки %d\n", WSAGetLastError());
		closesocket(NewConnection);
		WSACleanup();
		return -6;
	}
	// Створюємо з отриманих даних рядок мови С
	if (Ret <= 1023)
		DataBuffer[Ret] = '\0';
	else
	{
		printf("Повідомлення задовге!\n");
		return -7;
	}
	DataBuffer[Ret] = '\0';
	printf("Успішно отримано %d байтів в повідомленні \"%s\".\n", Ret, DataBuffer);

	//Передаємо повідомлення клієнту
	printf("Спроба передати повідомлення клієнту...\n");
	if ((Ret = send(NewConnection, msg, strlen(msg), 0)) == SOCKET_ERROR)
	{
		printf("Помилка send, номер помилки %d\n", WSAGetLastError());
		closesocket(NewConnection);
		WSACleanup();
		return -8;
	}
	printf("Успішно передано %d байт повідомлення \"%s\".\n", Ret, msg);
```

```
	printf("Закриваємо з'єднання з клієнтом.\n");
	closesocket(NewConnection);
	// Вивантажуємо Winsock
	WSACleanup();
	printf("Натисніть Enter для завершення.\n");
	getchar();
	return 0;
}
// TCP клієнт
#pragma comment(lib, "Ws2_32.lib")

#define _WINSOCK_DEPRECATED_NO_WARNINGS

#include <winsock2.h>
#include <stdio.h>
#include <windows.h>
#include <string.h>

int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	printf("TCP Клієнт \n");
	printf("___________\n\n");

	WSADATA wsaData;
	SOCKET s;
	struct sockaddr_in ServerAddr, ca;
	u_short Port = 5150;
	int Ret;
	char DataBuffer[1024];
	char* msg = (char*)"Привіт, я TCP клієнт!";
	int lmsg = strlen(msg);
	int caSize = sizeof(ca);

	// Ініціалізація Winsock версії 2.2
	if ((Ret = WSAStartup(MAKEWORD(2, 2), &wsaData)) != 0)
	{
		//Winsock не завантажився. Код помилки повертає функция WSAStartup
		printf("Помилка WSAStartup, номер помилки %d\n", Ret);
		return 1;
	}
	// Створимо новий сокет для з'єднання цього TCP клієнта
	s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
	if (s == INVALID_SOCKET)
	{
		printf("Помилка socket, номер помилки %d\n", WSAGetLastError());
		WSACleanup();
		return 2;
	}

	// Заповнимо у змінній ServerAddr адресну інформацію про сервер, до якого ми
хочемо підключитися.
	// Сервер слухає порт 5150, IP-адресу сервера вводимо з клавіатури
	char ipAddress[32];
	printf("ВВедіть  IP - адресу сервера : ");
	gets_s(ipAddress);
	ServerAddr.sin_family = AF_INET;
	ServerAddr.sin_port = htons(Port);
	ServerAddr.sin_addr.s_addr = inet_addr(ipAddress);

	// З'єднаємо сокет s з сервером
```

```
	printf("Спроба з'єднання з %s : %d ...\n", inet_ntoa(ServerAddr.sin_addr),
htons(ServerAddr.sin_port));
	if (connect(s, (struct sockaddr*)&ServerAddr, sizeof(ServerAddr)) ==
SOCKET_ERROR)
	{
		printf("Помилка connect, номер помилки %d\n", WSAGetLastError());
		closesocket(s);
		WSACleanup();
		return 3;
	}

	printf("З'єднання успішне.\n");

	// Отримуємо інформацію про  адресу та порт клієнта
	getsockname(s, (struct sockaddr*)&ca, &caSize);
	printf("Адреса і порт  клінта %s : %d \n", inet_ntoa(ca.sin_addr),
ntohs(ca.sin_port));

	// Тепер ми можемо обмінюватися даними з сервером через сокет s
	// Передаємо серверу повідомлення "Привіт, я TCP клієнт!"
	printf("Спроба передати повідомлення.\n");
	if ((Ret = send(s, msg, lmsg, 0)) == SOCKET_ERROR)
	{
		printf("Помилка send, номер помилки %d\n", WSAGetLastError());
		closesocket(s);
		WSACleanup();
		return 4;
	}
	printf("Успішно передано %d байт повідомлення \"%s\".\n", Ret, msg);

	// Отримуємо дані від сервера
	printf("Чекаємо дані для отримання...\n");
	if ((Ret = recv(s, DataBuffer, sizeof(DataBuffer), 0)) == SOCKET_ERROR)
	{
		printf("Помилка recv, номер помилки %d\n", WSAGetLastError());
		closesocket(s);
		WSACleanup();
		return -6;
	}
	// Створюємо з отриманих даних рядок мови С
	if (Ret <= 1023)
		DataBuffer[Ret] = '\0';
	else
	{
		printf("Повідомлення задовге!\n");
		return -7;
	}
	printf("Успішно отримано %d байт в повідомленні \"%s\".\n", Ret, DataBuffer);

	printf("Закриваємо з'єднання з сервером.\n");
	closesocket(s);
	// Вивантажуємо Winsock
	WSACleanup();
	printf("Натисніть Enter для завершення.\n");
	getchar();
	return 0;
}
```