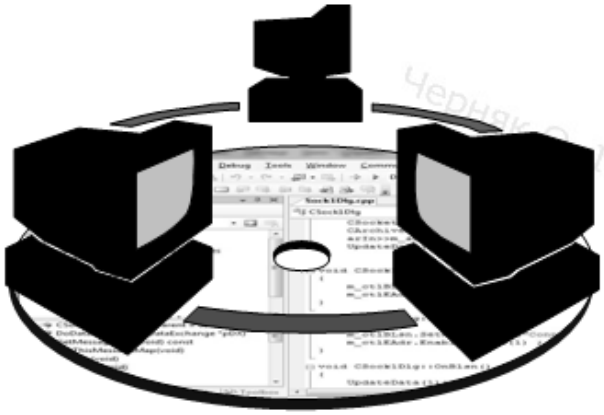
О. Д. Азаров
О. І. Черняк

Прикладне програмування у комп'ютерних мережах



Міністерство освіти і науки України
Вінницький національний технічний університет

О. Д. Азаров
О. І. Черняк

ПРИКЛАДНЕ ПРОГРАМУВАННЯ У КОМП'ЮТЕРНИХ МЕРЕЖАХ

Навчальний посібник

Вінниця
ВНТУ
2010

УДК 681.3.062(075)
ББК 32.973-01я73
А35

Рекомендовано до друку Вченою Радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 9 від 23.04.2009 р.)

Рецензенти:
В. П. Тарасенко, доктор технічних наук, професор НТУУ "КПІ", заслужений діяч науки і техніки України
Р. Н. Квєтний, доктор технічних наук, професор ВНТУ
І. Р. Арсенюк, кандидат технічних наук, доцент ВНТУ

Азаров, О. Д.
А35 Прикладне програмування у комп'ютерних мережах : навчальний посібник / О. Д. Азаров, О. І. Черняк. – Вінниця : ВНТУ, 2010. – 132 с.

У навчальному посібнику подано теоретичні та практичні відомості з прикладного програмування у комп'ютерних мережах мовою Visual C++ на основі використання WinAPI. Розглянуто такі технології розробки мережевих програм: перенаправлювач, поштові скриньки, іменовані канали, інтерфейс NetBIOS, сокети. Посібник орієнтований на програмування в Windows. Для вивчення матеріалу посібника потрібно мати знання з програмування мовою Visual C++. Посібник розроблений відповідно до плану кафедри та програми дисципліни "Прикладне програмування у комп'ютерних мережах".

УДК 681.3.062
ББК 32.973-01я73

ЗМІСТ

ПЕРЕДМОВА

Даний посібник призначений для вивчення дисципліни "Прикладне програмування у комп'ютерних мережах".

Дисципліна "Прикладне програмування у комп'ютерних мережах" вивчається на останньому курсі навчання студентами спеціалітету денної та заочної форм навчання, а також магістрами. Дисципліна базується на вивченні таких дисциплін: "Програмування", "Системне програмування", "Системне програмне забезпечення", "Візуальне програмування", "Основи комп'ютерних систем та мереж". Дисципліна є завершальною у підготовці спеціалістів зі спеціальності "Комп'ютерні системи та мережі".

У результаті вивчення дисципліни студенти повинні знати такі теоретичні питання: інформаційну структуру протоколів обміну даних, аспекти клієнт-серверної організації програмного забезпечення, аспекти роботи перенаправлювачів та поштових скриньок, аспекти обмеження доступу до ресурсів, аспекти роботи іменованих каналів, властивості і особливості інтерфейсу NetBIOS, принципи організації обміну даними за допомогою інтерфейсу прикладного програмування WINSOCK, аспекти використання адрес комп'ютерів та портів, види встановлення з'єднань між комп'ютерами та відповідні протоколи, технологію введення-виведення інформації у WINSOCK, мовні засоби, що застосовуються при створенні прикладних програм для роботи у мережах, переваги та недоліки технологій прикладного програмування у мережах.

Крім того, студенти повинні вміти створювати додатки для роботи у мережі на основі потокового та дайтаграмного передавання інформації з використанням перенаправлювачів, поштових скриньок, іменованих каналів, інтерфейсу прикладного програмування NetBIOS та сокетів. Студенти також повинні вміти створювати клієнт-серверні додатки з використанням синхронного і асинхронного режимів роботи функцій.

Для успішного вивчення матеріалу даного посібника необхідно мати знання принципів організації комп'ютерних мереж та техніки програмування мовою C++, а також мати практичні навички з програмування у середовищі Visual C++.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

У даному посібнику викладено теоретичні і практичні основи створення прикладних програм, що дозволяють обмінюватись інформацією між комп'ютерами, розташованими у мережі. В основу посібника покладено матеріали, подані у [1]. Всі програми у посібнику написані мовою Visual C++ і являють собою консольні додатки (*console applications*). Принципи будови операційної системи (*operation system*) WINDOWS та техніка створення програм для роботи в ній, знання яких необхідні для вивчення матеріалу посібника, описані в [2]. Принципи організації роботи комп'ютерних мереж (*computer network*) з використанням протоколу TCP/IP описані в [3]. Техніка програмування мовою Visual C++ описана в [4–7]. Для отримання практичних навичок з програмування у середовищі Visual C++ можна порекомендувати [8].

Посібник складається з п'яти розділів. У всіх розділах вивчаються технології мережевого програмування на основі бібліотеки WinAPI. Кожен розділ описує окрему технологію.

У першому розділі описано технологію перенаправлювача (*redirector*), що є компонентом операційної системи Windows і дозволяє додаткам обмінюватися інформацією у мережі за допомогою вбудованих служб файлової системи, так званої мережевої операційної системи (*network operating system, NOS*).

У другому розділі описано технологію поштових скриньок, що має клієнт-серверну архітектуру і використовує перенаправлювач. Технологія поштових скриньок реалізує простий однонаправлений механізм міжпроцесного зв'язку без встановлення з'єднання. Поштові скриньки дозволяють здійснювати передавання інформації від клієнта до сервера без забезпечення надійності передавання і цілісності даних.

У третьому розділі описано технологію іменованих каналів. Іменовані канали – це простий механізм зв'язку між процесами (*interprocess communication, IPC*), підтримуваний у Windows Vista, Windows XP, Windows NT, Windows 2000, Windows 98 (але не Windows CE). Іменовані канали забезпечують надійне одностороннє і двостороннє передавання даних між процесами на одному або на різних комп'ютерах у режимі потоку байтів (*byte stream*) або у режимі потоку повідомлень (*message stream*).

У четвертому розділі описано інтерфейс прикладного мережевого програмування NetBIOS. Спочатку даний інтерфейс був орієнтований на протокол NetBEUI, що фактично вже не використовується. Проте велика популярність NetBIOS привела до того, що була введена підтримка даного інтерфейсу і у протокол TCP/IP. Крім того, знати інтерфейс NetBIOS корисно ще й тому, що всі імена у мережі є іменами NetBIOS. Інтерфейс NetBIOS оснований на використанні єдиної функції Netbios, що має один параметр – вказівник на структуру типу NCB. Дана структура є складним типом і використовується для задання команд, режимів, параметрів і даних роботи функції Netbios.

Найбільший п'ятий розділ, що займає майже половину посібника, присвячено сучасному інтерфейсу програмування мережевих додатків (*network applications*) – Winsock. Winsock – це не протокол, а інтерфейс прикладного програмування мережевих взаємодій, реалізований на всіх платформах Win32. У даному розділі описано характеристики протоколів, що підтримуються Win32 і у свою чергу підтримують інтерфейс Winsock. Описано режими сокетів, алгоритм встановлення з'єднання та функції, необхідні для встановлення з'єднання і виконання обміну.

У кожному розділі наводяться приклади програм, що демонструють використання відповідних технологій та контрольні питання для самоперевірки.

Для самостійної роботи з вивчення мережевого програмування за допомогою даного посібника бажано мати доступ до комп'ютерної мережі і можливість виконувати програми хоча б на двох комп'ютерах у мережі, а також відповідний дозвіл або права адміністратора. Проте, можна виконувати мережеві програми і на одному комп'ютері. Для цього потрібно у програмах вказувати локальну адресу (*local address*). На комп'ютері повинна бути встановлена операційна система Windows 98 або більш високої версії. Крім того, повинно бути встановлене середовище програмування Visual Studio 6.0 або більш високої версії. Якщо на комп'ютері встановлені програми чи працюють служби, що обмежують можливість обміну інформацією у мережі, то потрібно зняти ці обмеження за допомогою встановлення відповідних параметрів або тимчасово відключити обмежувальні програмні засоби.

ВСТУП

Мережеве програмування є однією з центральних задач при розробці бізнес-додатків, оскільки потреба в ефективній і безпечній взаємодії різних комп'ютерів, що знаходяться в одній будівлі або розкидані по всьому світу, залишається основною для успішної діяльності більшості організацій.

З самого початку технології мережевого програмування були орієнтовані на використання у UNIX-подібних операційних системах. Широке використання операційної системи Windows і зростання ролі комп'ютерних мереж змусило Microsoft розробити інструментарій мережевого програмування (*network programming*) для своєї операційної системи. Слід відзначити, що свої засоби мережевого програмування Microsoft свідомо розробляла схожими на вже відомі в UNIX. Це розумна політика лідера інформаційного ринку, що дозволила значно спростити перенесення мережевих програм з однієї операційної системи на іншу.

У даному посібнику описані основні технології мережевого програмування, доступ до яких надає бібліотека WinAPI. Бібліотека WinAPI є основним інструментальним засобом Microsoft, призначеним для створення ефективних Windows-додатків. Всі технології більш високого рівня, такі як MFC, DOT NET базуються на даній бібліотеці. Тому освоєння принципів використання WinAPI дозволить не тільки створювати високоефективні додатки, але, також, краще розуміти концепцію і особливості мережевого програмування технологій більш високого рівня.

При розробці мережевих додатків Windows найчастіше використовуються сокети, проте для ефективної організації обміну інформацією між комп'ютерами потрібно вміти використовувати також інші програмні засоби мережевого програмування. У даному посібнику розглянуто основний набір таких засобів, що надається фірмою Microsoft у рамках середовища програмування Visual C++. Для обміну інформацією між комп'ютерами Microsoft надає такі основні засоби: перенаправлювач, поштові скриньки (*mailslots*), іменовані канали (*named pipes*), інтерфейс прикладного програмування NetBIOS, інтерфейс прикладного програмування WINSOCK. Кожен з цих засобів ефективний у відповідних випадках – саме тому Microsoft включила їх всіх до Visual Studio.

Перенаправлювач є основною складовою (*basic component*) набору

засобів Windows, що відповідає за роботу з дисками, файлами і папками на віддалених комп'ютерах. Перенаправлювач працює над транспортними протоколами і здійснює перенаправлення запитів файлової системи на віддалений комп'ютер, забезпечуючи при цьому досить ефективний механізм захисту від несанкціонованого доступу (*access protection*), оснований на дескрипторах безпеки. Це дозволяє додаткам використовувати для доступу до файлів через мережу і роботи з ними звичайні API-функції для роботи з файлами (типу CreateFile, ReadFile і WriteFile).

У даному розділі докладно розглядається використання перенаправлювача для передавання запитів введення-виведення (*input-output query*) на віддалені пристрої (*removed devices*) (саме на цьому заснований зв'язок у технологіях поштових скриньок і іменованих каналів). Перенаправлювач – найбільш простий механізм, ефективний у випадках, коли не потрібно підтримувати надійний обмін і забезпечувати цілісність переданої інформації.

Поштові скриньки використовують перенаправлювач і є засобом швидкого одностороннього обміну (*one-directional exchange*) інформацією за рахунок можливості записування її у спеціально створену область пам'яті віддаленого комп'ютера. Ця область і називається, власне, поштовою скринькою. Поштові скриньки реалізують клієнт-серверну архітектуру і дозволяють клієнтському процесу передавати повідомлення одному чи декільком серверним процесам. Вони також дозволяють передавати повідомлення між процесами на тому самому комп'ютері чи на різних комп'ютерах у мережі.

Розробка додатків, що використовують поштові скриньки, не вимагає знання мережевих транспортних протоколів (*network transport protocols*), таких як TCP/IP чи IPX. Оскільки поштові скриньки засновані на архітектурі широкомовлення (*broadcasting architecture*), вони не гарантують надійне передавання даних, але корисні, коли доставка даних не є життєво важливою.

Іменовані канали (*named pipes*) також використовують перенаправлювач. Вони надають можливість двостороннього як синхронного, так і асинхронного обміну інформацією. Розробка програм, що працюють з іменованими каналами, не є складною і не вимагає особливих знань механізму роботи основних мережевих протоколів

(network protocols) (таких як TCP/IP або IPX). Деталі роботи протоколів приховані від додатка, оскільки для обміну даними між процесами через мережу іменовані канали використовують перенаправлювач мережі Microsoft. До прикладів використання іменованих каналів можна віднести розробку системи керування даними, що дозволяє виконувати трансакції тільки певній групі користувачів.

Інтерфейс NetBIOS надає можливість швидкого і надійного обміну інформацією. Network Basic Input/Output System (NetBIOS) – стандартний інтерфейс прикладного програмування (*application programming interface, API*), розроблений Sytek Corporation для IBM у 1983 р. NetBIOS визначає програмний інтерфейс для мережевого зв'язку, але не обумовлює фізичний спосіб передавання даних мережею. У 1985 р. IBM спробувала сформувати цілісний протокол – створила NetBIOS Extended User Interface (Net-BEUI), інтегрований з інтерфейсом NetBIOS. Програмний інтерфейс NetBIOS незабаром набув такої популярності, що постачальники програмного забезпечення почали реалізовувати його для інших протоколів, таких як TCP/IP і IPX/SPX. У даний час NetBIOS використовують платформи і додатки в усьому світі, включаючи багато компонентів Windows Vista, Windows XP, Windows NT, Windows 2000 та Windows 98.

Сокети є сучасним широко розповсюдженим інструментом мережевого програмування, що забезпечує можливість двонаправленого надійного передавання інформації у режимі встановлення зв'язку або швидкого передавання інформації без встановлення зв'язку. Інтерфейс Winsock розроблено на основі Berceley Sockets (BSD) на платформах UNIX, що працює з багатьма мережевими протоколами. У середовищі Win32 даний інтерфейс став повністю незалежним від мережевих протоколів. За допомогою бібліотеки ws2-32.lib середовище програмування Visual C++ надає великий набір типів і функцій WinAPI для тонкого керування параметрами передавання інформації у мережі.

У посібнику розглянуті основні аспекти всіх перерахованих технологій мережевого програмування та наведені конкретні приклади програм, що реалізують дані технології.

1 ПЕРЕНАПРАВЛЮВАЧ

Операційна система Windows надає можливість додаткам обмінюватися інформацією мережею за допомогою вбудованих служб файлової системи, так званої мережевої операційної системи (*network operating system, NOS*).

Для обміну файлами, розташованими на локальному комп'ютері, додатки посилають запити на виконання введення-виведення операційній системі. Коли додаток відкриває файл, операційна система визначає пристрій, на якому знаходиться цей файл, і передає відповідний запит драйверу пристрою. Аналогічно здійснюється доступ до пристроїв мережею, тільки запит на роботу з файлами передається мережею пристрою на віддаленому комп'ютері. Це називається перенаправленням введення-виведення (*I/O redirection*).

Для доступу до файлів мережею додатки використовують звичайні API-функції для роботи з файлами (типу CreateFile, ReadFile і WriteFile).

1.1 Універсальні правила іменування

Імена UNC – це стандартний спосіб доступу до файлів і пристроїв з використанням віддаленої файлової системи. Такий підхід дозволяє додаткам не залежати від імен дисків і прозоро працювати з мережею. Імена UNC мають вигляд

\\сервер\ресурс\шлях

Перша частина – \\сервер, починається з двох обернених скісних рисок і імені віддаленого сервера, на якому знаходиться потрібний файл. Друга – \ресурс, це ім'я загального ресурсу, тобто папки у файловій системі, до якої відкритий загальний доступ користувачам мережі. Третя частина – \шлях, позначає шлях до потрібного файлу.

Припустимо, на сервері з ім'ям Myserver знаходиться папка D:\Myfiles\CoolMusic, надана для загального доступу під мережевим ім'ям Myshare, а у цій папці – файл Sample.mp3. Тоді для доступу до цього файлу з іншої машини треба вказати таке UNC-ім'я:

\\Myserver\Myshare\Sample.mp3

Усі деталі мережевого з'єднання організуються перенаправлювачем компонента доступу.

На рис. 1.1 зображені основні складові, що формують UNC-з'єднання у NOS в рамках Windows, а також показано, як передаються дані між компонентами клієнта і сервера NOS.

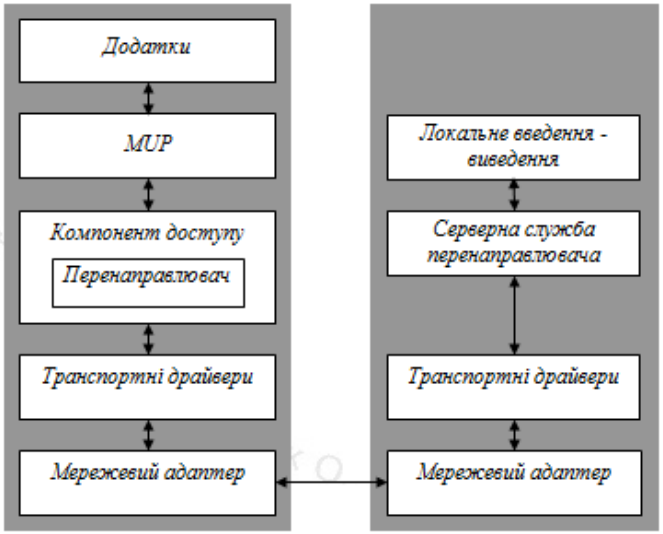


Рисунок 1.1 – Складові UNC-з'єднання

1.2 Постачальник декількох UNC

MUP – це вказівник ресурсу, відповідальний за вибір компонента доступу для обслуговування з'єднань за допомогою UNC-імен. Компонент мережевого доступу чи мережевий постачальник (*network provider*) – це служба, здатна використовувати мережеві пристрої для доступу до ресурсів, розташованих на віддаленому комп'ютері, наприклад, до файлів і принтерів. MUP використовує компонент мережевого доступу для організації зв'язку у відповідь на всі запити введення-виведення до файлів і принтерів через UNC-імена.

В операційній системі Windows може бути кілька компонентів доступу одночасно. Так, у платформу Windows вбудований клієнт для мереж Microsoft (Client for Microsoft Networks), але можна також встановити сторонні компоненти доступу, наприклад, клієнт для мереж

Novell. Таким чином, обслужити UNC-запит можуть кілька компонентів одночасно.

Головна функція MUP – вибір мережевого компонента, що повинен обслужити UNC-запит. MUP робить це, посилаючи паралельно UNC-імена з запиту усім встановленим компонентам доступу. Якщо який-небудь компонент відповідає, що здатен обслужити запит з даними іменами, то йому направляється весь запит. Якщо це можуть зробити кілька компонентів, то MUP вибирає той, що має найвищий пріоритет. Пріоритет компонентів визначається порядком, в якому вони були встановлені у системі. Цей пріоритет визначає параметр ProviderOrder у реєстрі Windows у розділі

\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\NetworkProvider\Order

1.3 Компонент мережевого доступу

Компоненти доступу повинні обслуговувати запити на з'єднання через UNC-імена. У Windows вони використовують для цього перенаправлювач.

У комплект Windows входить клієнт для мереж Microsoft (*Client for Microsoft Networks*), що раніше називався Microsoft Networking Provider (MSNP).

1.4 Призначення перенаправлювача

Перенаправлювач формує мережеві запити введення-виведенні відправляє їх серверній службі на віддаленому комп'ютері, що генерує локальні запити введення-виведення.

MSNP містить перенаправлювач, що прямо взаємодіє з рівнем мережевого транспорту і NetBIOS для забезпечення зв'язку між клієнтом і сервером. Додаткам, що його використовують, не потрібно знати деталі роботи мережевих протоколів: вони виконуються поверх TCP/IP, NetBEUI чи навіть IPX/SPX. Ця можливість дозволяє додаткам обмінюватися інформацією незалежно від фізичної організації мережі.

Утім, є один нюанс: щоб два додатки могли зв'язатися мережею, їхні робочі станції повинні мати хоча б один спільний протокол. Так, якщо на одному комп'ютері встановлений тільки TCP/IP, а на другому

комп'ютері – тільки IPX, перенаправлювач MSNP не зможе забезпечити зв'язок між ними.

Перенаправлювач MSNP зв'язується з іншими робочими станціями, посилаючи повідомлення серверній службі перенаправлювача на цих станціях. Такі повідомлення задають у стандартній структурі, що називається блок повідомлення сервера (*Server Message Block, SMB*). Протокол, яким перенаправлювач відправляє та одержує повідомлення з віддалених комп'ютерів, називається протоколом спільного використання файлів на основі блоків повідомлень сервера (*Server Message Block File Sharing Protocol*) чи просто протокол SMB.

1.5 Протокол SMB

Даний протокол був розроблений Microsoft і Intel наприкінці 80-х років для спрощення доступу додатків MS-DOS до віддалених файлових систем. Тепер він просто дозволяє перенаправлювачу MSNP у Windows використовувати структуру даних SMB при зв'язку зі службою сервера MSNP на віддаленій робочій станції. Структура даних SMB містить три основних компонента: код команди, параметри команди і дані користувача.

Протокол SMB базується на моделі запитів клієнта і відповідей сервера. Клієнт перенаправлювача MSNP створює структуру SMB із вказанням типу запиту у полі коду команди. Якщо команда вимагає відправлення даних (наприклад, SMB-команда Write), то вони додаються до запиту. Потім структура SMB відправляється транспортним протоколом (наприклад, TCP/IP) серверній службі на віддаленій робочій станції. Ця служба обробляє запит клієнта і повертає йому відповідну структуру SMB.

Тепер розглянемо приклад відкриття файлу \\Myserver\Myshare\Sample.mp3 мережею. При цьому відбувається така послідовність дій.

• Додаток направляє локальній ОС запит на відкриття цього файлу за допомогою API-функції CreateFile.

• Локальна ОС визначає через UNC-ім'я, що цей запит адресовано віддаленій машині з ім'ям \\Myserver, і передає його MUP.

• MUP визначає, що запит призначений компоненту доступу MSNP, оскільки саме цей компонент виявляє \\Myserver шляхом аналізу NetBIOS-імені.

• Запит введення-виведення передається перенаправлювачу MSNP.

• Перенаправлювач формує запит на відкриття файлу Sample.mp3 у віддаленій папці \Myshare як повідомлення SMB.

• Відформатоване повідомлення SMB передається мережевим транспортним протоколом.

• Сервер з ім'ям \\Myserver одержує SMB-запит мережею і передає його серверній службі свого перенаправлювача MSNP.

• Серверна служба виконує локальний запит введення-виведення на відкриття файлу Sample.mp3 у загальній папці \Myshare.

• Перенаправлювач сервера формує відповідне SMB-повідомлення з інформацією про успіх чи невдачу локального запиту на відкриття файлу.

• Відповідь сервера посилається мережевим транспортним протоколом назад клієнту.

• Перенаправлювач MSNP одержує відповідь SMB і передає код повернення локальній ОС.

• Локальна ОС передає код повернення у додаток, що викликав функцію CreateFile.

Перенаправлювач також керує доступом до ресурсів, що є однією з форм мережевої безпеки.

1.6 Безпека

При спробі додатка одержати доступ до таких ресурсів операційна система перевіряє, чи має цей додаток відповідні права. Основні види доступу: читання, записування і виконання. Windows керує доступом на основі дескрипторів безпеки (*security descriptors*) і маркерів доступу (*access tokens*).

1.6.1 Дескриптори безпеки

Усі захищені об'єкти мають дескриптор безпеки, що містить інформацію про порядок доступу до об'єкта. Дескриптор безпеки складається зі структури SECURITY_DESCRIPTOR і пов'язаної з ним інформації про безпеку, що містить:

• ідентифікатор безпеки (*Security Identifier, SID*) власника – визначає власника об'єкта;

• SID групи – визначає основну групу, в яку входить власник об'єкта;

• список керування доступом (*Discretionary Access Control List, DACL*) – вказує, хто і який тип доступу (читання, записування, виконання) має для даного об'єкта;

• системний список керування доступом (*System Access Control List, SACL*) – задає типи доступу до даного об'єкта, для яких генеруються записи у журнал аудита.

Додатки не можуть прямо змінювати вміст структури дескриптора безпеки. Утім, для цього можна використовувати API-інтерфейси безпеки Win32, що містять відповідні функції.

1.7 Списки і записи керування доступом

Поля DACL і SACL у дескрипторі безпеки – це списки керування доступом (ACL), що містять нуль чи більше записів керування доступом (*access control entities, ACE*). Кожна ACE керує доступом чи здійснює контроль за доступом до об'єкта певного користувача чи групи і містить:

• SID користувача чи групи, до яких застосовується ACE;

• маску, що задає права доступу (читання, записування, виконання);

• прапорець, що позначає тип ACE – дозвіл на доступ, заборона на доступ чи системний аудит.

Системний аудит застосовується тільки у списках SACL, а типи дозволу і заборона на доступ – у списках DACL. Якщо захищений об'єкт не має списку DACL (тобто, його DACL був обнулений API-функцією SetSecurityDescriptorDacl), то операційна система надає усім користувачам повний доступ до даного об'єкта. Якщо об'єкт має не порожній DACL, система надає тільки ті типи доступу, що явно зазначені у записах ACE даного DACL. Якщо у списку немає жодного запису ACE, система не надає нікому ніякого доступу. Якщо ж DACL містить кілька записів ACE, що дозволяють доступ, система неявно забороняє доступ усім користувачам і групам, яких немає у списку.

У більшості випадків задають тільки записи ACE, що дозволяють доступ, за винятком, якщо є запис, що дозволяє доступ до ресурсу усій групі, але потрібно заборонити доступ деяким членам цієї групи. При цьому запис ACE, що забороняє доступ, повинен обов'язково передувати запису, що відкриває доступ усій групі, оскільки система читає записи один за одним і припиняє читання, з'ясувавши, що доступ даному

користувачу відкрито чи закрито. Тобто, якщо запис, що дозволяє доступ групі, буде першим, система прочитає його і надасть доступ неповноважному користувачу зі складу групи.

На рис. 1.2 показано список DACL, що надає доступ для читання групі Net Team. Ця група складається з користувачів Anthony, Jim і Gary, і потрібно надати право читання усім, крім Anthony. Тому запис, що забороняє доступ користувачу Anthony, передує запису, що дозволяє доступ групі Net Team. На даному рисунку також показано запис ACE, що надає користувачу Jim доступ для записування.

Нагадаємо, що додатки не можуть прямо працювати з ACE, а використовують для цього спеціальні API-інтерфейси.

1.8 Маркери доступу

При вході користувача в операційну систему Windows перевіряються його реквізити: ім'я облікового запису і пароль. Якщо вхід дозволено, система створює маркер доступу і заносить у нього SID користувача. Кожен процес, запущений від імені цього користувача, одержить копію маркера. При спробі доступу процесу до захищеного об'єкта SID у маркері доступу процесу порівнюється з SID у списках DACL об'єкта для визначення прав доступу.

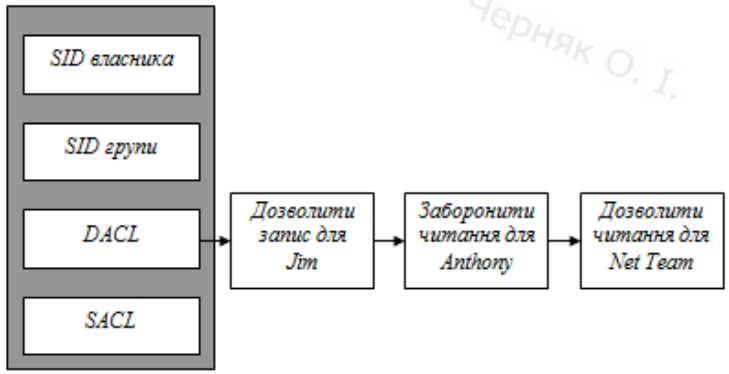


Рисунок 1.2 – Об'єкт файлу з відповідним йому DACL

1.9 Реквізити безпеки

Розглянемо тепер обмеження доступу до об'єктів мережею. Для цього перенаправлювач MSNP встановлює безпечне з'єднання клієнт-сервер, створюючи реквізити сеансу користувача. На рисунку 1.3 зображено схему застосування реквізитів безпеки у локальній мережі, що має контролер домену, при звертанні робочої станції А до ресурсу робочої станції В.

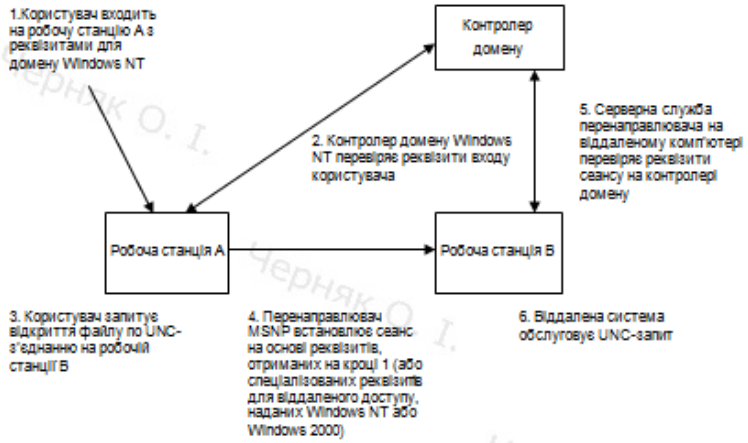


Рисунок 1.3 – Застосування реквізитів безпеки

Реквізити користувача бувають двох типів: основні реквізити входу (*primary login*) і реквізити сеансу (*session login*). Коли користувач реєструється на робочій станції, вводяться ім'я і пароль, що стають його основними реквізитами входу і заносяться у маркер доступу. В один момент часу користувач може мати лише один набір реквізитів.

При спробі доступу до віддаленого ресурсу реквізити входу користувача використовуються для перевірки прав доступу до цього ресурсу. Якщо доступ дозволено, перенаправлювач MSNP створює сеанс зв'язку між комп'ютером користувача та віддаленим ресурсом і надає сеансу реквізити користувача.

Кожен сеанс зв'язку може мати тільки один набір реквізитів.

На віддаленому сервері безпеку контролює серверна служба перенаправлювача MSNP. При спробі одержати доступ до захищеного об'єкта, вона використовує реквізити сеансу для створення маркера віддаленого доступу. Далі обмеження доступу до захищених об'єктів забезпечується так само, як і при локальному доступі.

1.10 Приклади

Додатки Win32 можуть використовувати API-функції CreateFile, ReadFile і WriteFile для створення, відкриття і обміну файлами через мережу з використанням перенаправлювача MSNP. У лістингу 1.1 наведено приклад коду додатка, що створює файл за допомогою UNC-з'єднання. Тут і далі символом ⏎ позначається перенесення команди у наступний рядок.

Лістинг 1.1 – *Простий приклад створення файлу*

```
#include «windows.h>
#include <stdio.h>
void main(void)
{
    HANDLE FileHandle;
    DWORD BytesWritten;
    // Відкриття файлу \\Myserver\Myshare\Sample.txt
    if ((FileHandle = CreateFile("\\\\MyServer\\Myshare\\Sample.txt", ⏎
    GENERIC_WRITE | GENERIC_READ, FILE_SHARE_READ | ⏎
    FILE_SHARE_WRITE, NULL,CREATE_ALWAYS, ⏎
    FILE_ATTRIBUTE_NORMAL, NULL))== ⏎
    INVALID_HANDLE_VALUE)
    {
        printf("CreateFile failed with error %d\n", GetLastError());
        return;
    }
    // Записування 14 байтів у новий файл
    if (WriteFile(FileHandle, "This is a test", 14,&BytesWritten, NULL) ⏎
    == 0)
    {
```

```
        printf("WriteFile failed with error %d\n", GetLastError());
        return;
    }
    if (CloseHandle(FileHandle) == 0)
    {
        printf("CloseHandle failed with error %d\n", GetLastError());
        return;
    }
}
```

1.11 Контрольні питання

1. Що таке UNC? Опишіть вигляд імені UNC.
2. Чим регламентується формат обміну інформацією за допомогою перенаправлювача?
3. Опишіть структуру компонент при обміні інформацією за допомогою перенаправлювача.
4. Що таке MUP, MSNP, SMB?
5. Опишіть послідовність дій, що виконуються при відкритті файлу на віддаленому комп'ютері?
6. Як відбувається керування доступом до окремого файлу чи папки?
7. Опишіть типи реквізитів користувача.
8. Яка роль списку SACL?
9. Яка роль списку ACL?
10. За допомогою яких функцій здійснюється створення файлів на віддаленому комп'ютері і запис у них інформації? Опишіть параметри цих функцій.
11. Напишіть програму записування інформації у файл на віддаленому комп'ютері за допомогою перенаправлювача.
12. Опишіть схему застосування реквізитів безпеки.
13. Опишіть структуру ACE.
14. Яким чином прикладна програма може змінити запис ACE?
15. У яких випадках використовуються дозволяючі записи в ACE, а у яких – забороняючі?

2 ПОШТОВІ СКРИНЬКИ

У Microsoft Windows реалізовано простий однонаправлений механізм міжпроцесного зв'язку (*interpocess communication, IPC*), названий поштовими скриньками (*mailslots*). Поштові скриньки розташовуються в оперативній пам'яті іменованій, як Mailslot. Основне обмеження поштових скриньок – вони допускають тільки ненадійне односпрямоване передавання даних від клієнта до сервера. Основна перевага – клієнтські додатки можуть легко посилати широкомовні повідомлення одному чи декільком серверним додаткам.

2.1 Подробиці впровадження поштових скриньок

Клієнтський і серверний додатки використовують стандартні функції введення-виведення файлової системи Win32 (такі як ReadFile і WriteFile) для відправлення й одержання даних поштовою скринькою, а також правила іменування файлової системи Win32. Для створення й ідентифікації поштових скриньок перенаправлювач Windows використовує файлову систему Mailslot File System (MSFS).

2.2 Імена поштових скриньок

Поштові скриньки іменуються за таким правилом:

\\сервер\Mailslot\[шлях]ім'я

\\сервер – ім'я сервера, на якому створюється поштова скринька і виконується серверний додаток;

\\Mailslot – фіксоване обов'язкове ім'я; \[шлях]ім'я – дозволяє додаткам унікальним чином визначати й ідентифікувати ім'я поштової скриньки.

\[шлях]ім'я – дозволяється задавати кілька рівнів каталогів. Наприклад, дозволяються такі імена поштових скриньок:

\\Oreo\Mailslot\Mymailslot

\\Testserver\Mailslot\Cooldirectory\Funtest\Anotherfflallslot

\\.\Mailslot\Easymailslot

**\Mailslot\Myslot*

Поле \\server може являти собою крапку (.), зірочку (*), ім'я домену чи сервера.

Повідомлення, як правило, пересилаються без встановлення з'єднання, але залежно від розміру повідомлення перенаправлювач Windows може встановлювати з'єднання на комп'ютерах.

2.3 Розміри повідомлень

Для передавання повідомлень через мережу поштові скриньки, як правило, використовують дейтаграми (*datagram*) – невеликі порції даних, що передаються без встановлення з'єднання, якщо розмір повідомлення перевищує 424 байти. Повідомлення розміром більше 426 байтів передаються з використанням протоколу, що вимагає встановлення з'єднання. Однак при встановленні з'єднання допускається з'єднання тільки одного клієнта з одним сервером.

Навіть при встановленні з'єднання інтерфейс поштової скриньки не гарантує, що повідомлення буде дійсно записано у поштову скриньку. Наприклад, при відправленні повідомлення від клієнта неіснуючому серверу, інтерфейс поштової скриньки не повідомить клієнтському додатку, що не зміг передати дані. Щоб забезпечити повну сумісність усіх платформ Windows, потрібно обмежити розмір повідомлень значенням 424 байтами. При необхідності встановлення з'єднання замість поштових скриньок використовують іменовані канали.

2.4 Компіляція додатка

При розробці програми клієнта чи сервера поштових скриньок у Microsoft Visual C++ необхідно включати у програмні файли додатків заголовний файл Winbase.h. Якщо у додаток включено файл Windows.h, то Winbase.h можна опустити. Додаток також повинен компонуватися з бібліотекою Kernel32.1ib.

2.5 Коди помилок

Усі API-функції Win32, що використовуються при розробці клієнтів і серверів поштових скриньок (за винятком CreateFile і CreateMailslot), у випадку невдалого виконання повертають нуль. API-функції CreateFile і CreateMailslot повертають значення INVALID_HANDLE_VALUE. Для одержання кодів помилок цих функцій додаток повинен викликати

функцію GetLastError. Повний список кодів помилок наведено у файлі Winerror.h.

2.6 Загальні дані про архітектуру клієнт-сервер

Поштові скриньки використовують просту архітектуру клієнт-сервер, у якій дані передаються від клієнта до сервера односпрямовано. Сервер відповідає за створення поштової скриньки і є єдиним процесом, що може читати з нього дані. Клієнти поштових скриньок – це процеси, що відкривають екземпляри поштових скриньок і єдині, що мають право записувати у них дані.

2.7 Сервер поштових скриньок

Для реалізації поштової скриньки потрібно написати серверний додаток, у якому виконати такі дії.

1. Створити описувач поштової скриньки за допомогою API-функції CreateMailslot.
2. Одержати дані від будь-якого клієнта шляхом виклику API-функції ReadFile з описувачем поштової скриньки як параметром.
3. Закрити описувач поштової скриньки за допомогою API-функції CloseHandle.

Серверні процеси створюють поштові скриньки за допомогою виклику API-функції ***CreateMailslot***, що визначена так:

HANDLE CreateMailslot(
LPCTSTR lpName,
DWORD nMaxMessageSize,
DWORD lReadTimeout,
LPSECURITY_ATTRIBUTES lpSecurityAttributes);

lpName – задає ім'я поштової скриньки. Воно повинно мати такий вигляд:

\\.\Mailslot\[шлях]ім'я

Тут ім'я сервера замінене крапкою, що означає локальний комп'ютер. Це є обов'язковим, оскільки не можна створити поштову скриньку на віддаленому комп'ютері. Ім'я повинно бути унікальним, але може бути простим чи включати повний шлях.

nMaxMessageSize – задає максимальний розмір (у байтах)

повідомлення, що може бути записане у поштову скриньку. Якщо клієнт записує повідомлення більшого розміру, сервер не бачить це повідомлення. Якщо задати значення нуль, то сервер буде приймати повідомлення будь-якого розміру.

lReadTimeout – задає кількість часу (у мілісекундах), протягом якого операції читання чекають вхідних повідомлень. Значення MAILSLOT_WAIT_FOREVER блокує читання, доки вхідні дані не стануть доступні. Якщо задати нульове значення, операції читання завершуються негайно.

lpSecurityAttibutes – задає права доступу до поштової скриньки. Він повинен мати значення NULL, тому що у Windows система безпеки до об'єктів не використовується.

Поштова скринька відносно безпечна тільки при локальному введенні-виведенні. Коли клієнт намагається відкрити цю скриньку, він використовує крапку (.) як ім'я сервера. Клієнт може обійти систему безпеки, задавши замість крапки фактичне ім'я сервера, як при віддаленому виклику введення-виведення. Параметр lpSecurityAttibutes не реалізований для віддаленого введення-виведення. Таким чином, поштові скриньки тільки частково відповідають моделі безпеки Windows, реалізованій у стандартних файлових системах. Тому будь-який клієнт поштової скриньки у мережі може відправляти дані серверу.

Сервер – єдиний процес, що може читати дані з поштової скриньки. Для цього він повинен викликати Win32-функцію ***ReadFile***, визначену так:

```
BOOL ReadFile(
HANDLE hFile.
LPVOID lpBuffer,
DWORD nNumberOfBytesToRead,
LPDWORD lpNumberOfBytesRead,
LPOVERLAPPED lpOverlapped);
```

lpBuffer і nNum-berOfBytesToRead – визначають, скільки даних може бути зчитано з поштової скриньки. Важливо задати розмір буфера більшим, ніж параметр nMaxMessageSize з API-виклику CreateMailslot. Крім того, розмір буфера повинен перевищувати розміри вхідних повідомлень, інакше ReadFile видасть код помилки ERROR_INSUFFICIET_BUFFER.

lpNumberOfBytes – повертає кількість зчитаних байтів після

завершення роботи ReadFile.

lpOverlapped – дозволяє зчитувати дані з поштової скриньки асинхронно. Він використовує перекрите введення-виведення. За замовчуванням, виконання ReadFile блокує введення-виведення і очікує, доки дані не стануть доступні для читання. У лістингу 2.1 подано приклад написання простого сервера поштових скриньок.

Лістинг 2.1 – Приклад сервера поштових скриньок

```
// Served.cpp
//
#include <windows.h>
#include <stdio.h>
void main(void)
{
    HANDLE Mailslot;
    char buffer[256];
    DWORD NumberOfBytesRead;
    // Створення поштової скриньки
    if ((Mailslot = CreateMailslot("\\\\.\\Mailslot\\Myslot", 0,
    MAILSLOT_WAIT_FOREVER, NULL)) ==
    INVALID_HANDLE_VALUE)
    {
        printf("Failed to create a mailslot %d\n", GetLastError());
        return;
    }
    // Нескінченне читання даних з поштової скриньки
    while(ReadFile(Mailslot, buffer, 256, &NumberOfBytesRead, NULL)
    != 0)
    {
        buffer[NumberOfBytesRead]=0;
        printf("%s\n", buffer);
    }
    CloseHandle(Mailslot);
}
```

2.8 Клієнт поштових скриньок

Для реалізації клієнта потрібно розробити додаток, що посилається на існуючу поштову скриньку і записує у неї дані. У клієнтському додатку необхідно виконати такі кроки.

1. Відкрити описувач-посилання на поштову скриньку за допомогою API-функції CreateFile.

2. Записати дані у поштову скриньку, викликавши API-функцію WriteFile.

3. Закрити описувач поштової скриньки за допомогою API-функції CloseHandle.

Коли клієнт відкриває описувач-посилання на поштову скриньку, вона не встановлює зв'язок із сервером поштової скриньки. На поштові скриньки посилаються шляхом виклику API-функції ***CreateFile***, визначеної так:

HANDLE CreateFile(
LPCTSTR lpFileName,
DWORD dwDesiredAccess,
DWORD dwShareMode,
LPSECURITY_ATRIBUTES lpSecurityAttributes,
DWORD dwCreationDisposition,
DWORD dwFlagsAndAttributes,
HANDLE hTemplateFile);

lpFileName – ім'я поштової скриньки. Правила іменування поштових скриньок такі:

\\.\mailslot\ім'я – визначає локальну поштову скриньку на тому ж комп'ютері;

\\ім'я_сервера\mailslot\ім'я – визначає віддалений сервер поштової скриньки з ім'ям ім'я_сервера;

\\ім'я_домену\mailslot\ім'я – визначає всі поштові скриньки з іменем ім'я у домені ім'я_домену;

*\mailslot\ім'я – визначає всі поштові скриньки з іменем ім'я в основному домені системи.

dwDesiredAccess – повинен мати значення GENERIC_WRITE, тому що клієнт може тільки записувати дані на сервер.

dwShareMode – повинен мати значення FILE_SHARE_READ,

дозволяючи серверу відкривати і виконувати операції читання з поштової скриньки.

lpSecurityAttibutes – не впливає на поштові скриньки, йому потрібно задати значення NULL.

dwCreationDisposition – повинен дорівнювати OPEN_EXISTING.

dwCreationDisposition – не має значення, якщо сервер працює віддалено.

dwFlagsAndAttributes – повинен мати значення FILE_ATTRIBUTE_NORMAL, а hTemplateFile – значення NULL. Якщо сервер не створив поштову скриньку, API-функція CreateFile поверне помилку.

Після успішного створення описувача можна записувати дані у поштову скриньку. Клієнт може тільки записувати дані у поштову скриньку за допомогою Win32-функції ***WriteFile***:

BOOL WriteFile(
HANDLE hFile,
LPCVOID lpBuffer,
DWORD nNumberOfBytesToWrite,
LPDWORD lpNumberOfBytesWritten,
LPOVERLAPPED lpOverlapped);.

hFile – описувач-посилання, що повертається функцією Create File. lpBuffer і nNumberOfBytesToWrite – визначають, скільки байтів буде відправлено від клієнта серверу. Максимальний розмір повідомлення – 64 Кбайти. Якщо описувач поштової скриньки створений з іменем домену чи з зірочкою, розмір повідомлення не повинен перевищувати 424 байти. Якщо клієнт спробує відправити повідомлення більшого розміру, функція WriteFile поверне помилку ERROR_BAD_NETPATH (щоб дізнатися код помилки, потрібно викликати функцію GetLastError). Це відбувається тому, що повідомлення посилається усім серверам мережі як широкомовна дейтаграма.

lpNumberOfBytesWritten – повертає кількість байтів, відправлених серверу після завершення функції WriteFile.

lpOverlapped – дозволяє записувати дані у поштову скриньку в асинхронному режимі. Оскільки поштові скриньки обмінюються даними без встановлення з'єднання, функція WriteFile не блокує введення-виведення. У програмі клієнта цей параметр повинен дорівнювати NULL.

У лістингу 2.2 наведено приклад простого клієнта поштової скриньки.

Лістинг 2.2 – Приклад клієнта поштової скриньки

```
// Client.cpp
#include <windows.h>
#include <stdio.h>
void main(int argc, char *argv[])
{
	HANDLE Mailslot;
	DWORD BytesWritten;
	char ServerName[256];
	// Перевірка наявності у командному рядку імені сервера
	if (argc < 2)
	{
		printf("Usage: client <server name>\n");
		return;
	}
	sprintf(ServerName,"\\\\%s\\Mailslot\\Myslot", argv[1]);
	if ((Mailslot = CreateFile(ServerName, GENERIC_WRITE,
	FILE_SHARE_READ, NULL, OPEN_EXISTING,
	FILE_ATTRIBUTE_NORMAL,
	NULL)) == INVALID_HANDLE_VALUE)
	{
		printf("CreateFile failed with error %d\n", GetLastError());
		return;
	}
	if (WriteFile(Mailslot, "This is a test", 14, &BytesWritten, NULL) == 0)
	{
		printf("WriteFile failed with error %d\n", GetLastError());
		return;
	}
	printf("Wrote %d bytes\n", BytesWritten);
	CloseHandle(Mailslot);
}
```


2.9 Додаткові API-функції поштових скриньок

Сервер поштової скриньки може використовувати дві додаткові функції для взаємодії з поштовою скринькою: GetMailslotInfo і SetMailslotInfo. GetMailslotInfo повертає розмір повідомлення, коли воно прибуває у поштову скриньку, що дозволяє динамічно настроювати буфери для вхідних повідомлень змінної довжини. ***GetMailslotInfo*** може також застосовуватися для опитування наявності вхідних даних:

```
BOOL GetMailslotInfo(
HANDLE hMailslot,
LPDWORD lpMaxMessageSIze,
LPDWORD lpNextSize,
LPDWORD lpMessageCount,
LPDWORD lpReadTimeout);
```

hMailslot – вказівник на поштову скриньку, повернутий викликом функції CreateMailslot.

lpMaxMessageSize – задає розмір повідомлення (у байтах), що можна записати у поштову скриньку.

lpNextSize – вказує на розмір такого повідомлення (у байтах).

lpMesssageCount – задає буфер, куди записується загальна кількість повідомлень, що очікують прочитання. Цей параметр також задають для перевірки наявності даних.

lpReadTimeout – вказує на буфер, що повертає час тайм-ауту (у мілісекундах), протягом якого операція читання чекає записування повідомлення у поштову скриньку.

Повернення функцією значення MAILSLOT_NO_MESSAGE, вказує, що у даний момент поштова скринька не має даних. Сервер використовує цей параметр для неблокуючої перевірки наявності вхідних даних. Але краще для цього, використовувати перекрите введення-виведення Win32.

Функція ***SetMailslotInfo*** задає значення тайм-ауту для поштової скриньки, протягом якого операція читання очікує вхідних повідомлень. Таким чином, додаток може змінити спосіб читання від блокуючого до неблокуючого чи навпаки. Функція SetMailslotInfo визначена так:

```
BOOL SetMailslotInfo( HANDLE hMailslot, DWORD ReadTimeout);
```

hMailslot – вказівник на поштову скриньку, повернутий викликом

API-функції CreateMailslot.

ReadTimeout – визначає кількість часу (у мілісекундах), протягом якого операція читання очікує записування повідомлення у поштову скриньку. Якщо воно дорівнює нулю, то під час відсутності повідомлень операції читання повертаються негайно, якщо MAILSLOT_WAIT_FOREVER – будуть чекати нескінченно довго.

2.10 Контрольні питання

1. Опишіть обмеження на розмір повідомлень для поштових скриньок та напрямок обміну інформацією за допомогою поштових скриньок.
2. Які дії повинна виконувати програма, що є сервером поштової скриньки?
3. Наведіть приклади іменування поштових скриньок.
4. Яка функція записує дані у поштову скриньку? Опишіть її параметри.
5. Напишіть програму сервера поштової скриньки.
6. За допомогою якої функції можна встановити, чи є у поштовій скриньці вхідні повідомлення? Опишіть параметри даної функції.
7. Які дії повинна виконувати програма, що є клієнтом поштової скриньки?
8. Напишіть програму клієнта поштової скриньки.
9. За допомогою якої функції можна встановити для поштової скриньки час очікування вхідних повідомлень? Опишіть параметри даної функції.
10. За допомогою якої функції задається час очікування поштовою скринькою вхідних повідомлень? Опишіть параметри даної функції.
11. За допомогою якої функції сервер створює поштову скриньку? Опишіть параметри даної функції.
12. За допомогою якої функції клієнт отримує доступ до поштової скриньки? Опишіть параметри даної функції.
13. За допомогою яких функцій клієнт і сервер поштової скриньки обмінюються інформацією? Опишіть параметри даних функцій.

3 ІМЕНОВАНІ КАНАЛИ

Іменовані канали – це простий механізм зв'язку між процесами, підтримуваний у Microsoft Windows. Іменовані канали забезпечують надійне односторонне і двосторонне передавання даних між процесами на одному або різних комп'ютерах. Розробка програм, що працюють з іменованими каналами, є простою і не вимагає особливих знань механізму роботи мережевих протоколів. Дуже важливим є те, що іменовані канали дозволяють скористатися вбудованими можливостями захисту наданими конкретними реалізаціями операційної системи Windows.

До прикладів використання іменованих каналів можна віднести розробку системи керування даними, що дозволяє виконувати трансакції тільки певній групі користувачів. Розглянемо таку ситуацію: в офісі є комп'ютер з конфіденційною інформацією, доступ до якої повинен мати тільки управлінський персонал. Кожен співробітник фірми може бачити цей комп'ютер зі своєї робочої станції та використовувати його загальнодоступні дані, проте прості службовці не мають права доступу до конфіденційних даних. Дана проблема вирішується за допомогою іменованих каналів. Можна розробити серверний додаток, що виконуватиме транзакції над конфіденційними даними залежно від запитів клієнтів.

Іменовані канали основані на архітектурі клієнт-сервер, що забезпечує надійне передавання даних.

3.1 Деталі реалізації іменованих каналів

Для роботи з файловою системою Windows *(File System, FS)* іменовані канали використовують інтерфейс файлової системи іменованих каналів (*Named Pipe File System, NPFS*). Для отримання і відправлення даних сервер і клієнт викликають стандартні API-функції Win32 читання і записування, такі як ReadFile і WriteFile. Ці функції дозволяють додаткам використовувати звичайні правила іменування файлової системи Win32, а також використовувати можливості захисту, що надає певна версія операційної системи Windows. Відправлення і отримання даних через мережу здійснюється за допомогою інтерфейсу NPFS, що використовує перенаправлювач MSNP. Це робить його незалежним від протоколу при

розробці додатків. Назви іменованих каналів повинні відповідати формату Universal Naming Convention (UNC).

3.2 Правила іменування каналів

Імена каналів мають такий формат

\\server\Pipe\ [шлях]ім'я

Даний рядок складається з трьох частин \\сервер, \Pipe і \[шлях]ім'я, де \\сервер – ім'я сервера, на якому створений іменований канал \Pipe – фіксований обов'язковий рядок, що повідомляє систему про належність до NPFS [шлях]ім'я – унікальне ім'я каналу, що включає декілька рівнів каталогів. Ось приклади правильних назв іменованих каналів

\\myserver\PIPE\mypipe

\\Testserver\pipe\cooldirectory\funtest\jim

\\.\Pipe\Easynamedpipe

Ім'я сервера може бути представлене крапкою.

3.3 Режими: побайтовий і повідомлень

Іменовані канали використовують два режими передавання даних – побайтовий режим і режим повідомлень. У першому режимі повідомлення передаються безперервним потоком байтів між клієнтом і сервером. Це означає, що клієнт і сервер точно не знають, скільки байтів прочитується або записується у канал у певний момент часу. Таким чином, запис однієї кількості байтів за одну операцію з одного боку каналу не означає читання тієї ж кількості байтів за одну операцію з іншого боку. У другому режимі клієнт і сервер відправляють і приймають дані дискретними блоками, при цьому кожне повідомлення прочитується повністю. На рис. 3.1 зображена схема обміну інформацією в обох режимах.

3.4 Компіляція додатків

При створенні клієнта або сервера іменованого каналу за допомогою Microsoft Visual C++ необхідно включити у програмні файли заголовний файл Winbase.h. Якщо додаток включає файл Windows.h (як правило, це так), файл Winbase.h можна опустити. Крім того, як уже згадувалося, необхідно підключити бібліотеку Kernel32.1ib.

3.5 Коди помилок

Всі API-функції Win32 (окрім CreateFile і CreateNamedPipe), що використовуються при розробці сервера і клієнта іменованого каналу, при виникненні помилки повертають нуль. Функції CreateFile і CreateNamedPipe повертає значення INVALID_HANDLE_VALUE. Докладну інформацію про помилку можна отримати за допомогою функції GetLastError. Повний список кодів помилок – у файлі Winerror.h.

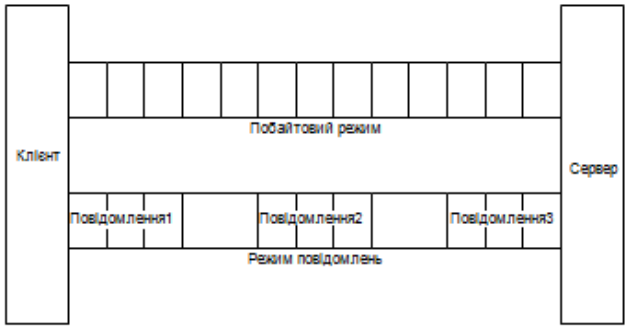


Рисунок 3.1 – Режими: побайтовий і повідомлень

3.6 Архітектура іменованих каналів

Іменовані канали мають просту архітектуру клієнт-сервер, при якій дані між віддаленими комп'ютерами передаються в одному або в обох напрямках. Останнє дозволяє і серверу, і клієнту відправляти та приймати дані. Основна відмінність сервера від клієнта полягає в тому, що тільки сервер може створити іменований канал і прийняти з'єднання з клієнтом. Клієнтський додаток встановлює з'єднання з існуючим сервером. Після цього сервер і клієнт можуть читати і записувати дані у канал за допомогою стандартних API-функцій Win32, таких як ReadFile і WriteFile. Сервер іменованого каналу працює тільки на комп'ютері з Windows XP, Windows Vista, Windows NT або Windows 2000. Windows 98 не підтримує створення іменованих каналів. Це обмеження не дозволяє двом комп'ютерам Windows 98 безпосередньо обмінюватися даними, проте їх клієнти можуть підключатися до серверів на комп'ютерах Windows NT і Windows 2000.

3.7 Деталі реалізації сервера

Сервер іменованого каналу створює екземпляри каналів, до яких підключаються клієнти. Для сервера екземпляр іменованого каналу – це просто описувач, за допомогою якого встановлюється з'єднання з локальними або віддаленими клієнтами. Процес створення простого сервера полягає у послідовному використанні API-функцій:

- CreateNamedPipe – для створення екземпляра іменованого каналу;
- ConnectNamedPipe – для прослуховування клієнтських з'єднань;
- ReadFile і WriteFile – для отримання і відправлення даних;
- DisconnectNamedPipe – для завершення з'єднання;
- CloseHandle – для закриття описувача іменованого каналу.

Спочатку сервер повинен створити екземпляр іменованого каналу за допомогою API-функції ***CreateNamedPipe***:

HANDLE CreateNamedPipe(
LPCTSTR lpName,
DWORD dwOpenMode,
DWORD dwPipeMode,
DWORD nMaxInstances,
DWORD nOutBufferSize,
DWORD nInBufferSize,
DWORD nDefaultTimeOut,
LPSECURITY_ATTRIBUTES lpSecurityAttributes);

lpName – визначає назву іменованого каналу, що відповідає формату UNC

\\.\Pipe\[path]\name

Ім'я сервера подане крапкою – це означає, що сервером є локальний комп'ютер. Іменований канал не можна створити на віддаленому комп'ютері. Частина параметра [шлях]\ім'я визначає унікальне ім'я каналу. Це може бути просто ім'я файлу або повний шлях до нього.

dwOpenMode – визначає напрямок передавання, керування введенням-виведенням і безпеку каналу. У табл. 3.1 перераховані прапорці, комбінації яких використовують при створенні каналу. Прапорці PIPE_ACCESS_ визначають напрямок передавання даних між сервером і клієнтом. Якщо відкрити канал з прапорцем PIPE_ACCESS_DUPLEX, передавання даних через цей канал буде двонаправленим.

Таблиця 3.1 – Прапорці режимів створення іменованого каналу

Режим відкриття	Прапорець	Опис
Напрямок	PIPE_ACCESS_DUPLEX	Канал двонаправлений: серверні і клієнтські процеси можуть приймати і відправляти дані через канал
	PIPE_ACCESS_OUTBOUND	Дані передаються тільки від сервера до клієнта
	PIPE_ACCESS_INBOUND	Дані передаються тільки від клієнта до сервера
Керування введенням-виведенням	FILE_FLAG_WRITE_THROUGH	Функції записування не повертають значення, доки дані передаються мережею або знаходяться у буфері віддаленого комп'ютера. Застосовується тільки для іменованих каналів, що працюють у побайтовому режимі
	FILE_FLAG_OVERLAPPED	Дозволяє використовувати перекрите введення-виведення при виконанні операцій читання, записування і з'єднання
Безпека	WRITE_DACL	Дозволяє додатку змінювати список DACL іменованого каналу
	ACCESS_SYSTEM_SECURITY	Дозволяє додатку змінювати список SACL іменованого каналу
	WRITE_OWNER	Дозволяє додатку змінювати власника іменованого каналу і групового SID

При відкритті каналу з прапорцями PIPE_ACCESS_INBOUND або PIPE_ACCESS_OUTBOUND дані передаватимуться тільки від клієнта серверу або навпаки. На рис. 3.2 зображені комбінації прапорців і напрямки передавання між сервером і клієнтом в операціях введення-виведення що виконуються сервером. При застосуванні прапорця FILE_FLAG_WRITE_THROUGH функції записування не повертають значення, доки дані не будуть передані через мережу або не з'являться у

буфері віддаленого комп'ютера. Цей прапорець використовують, коли сервер і клієнт знаходяться на різних комп'ютерах і іменований канал працює у побайтовому режимі.

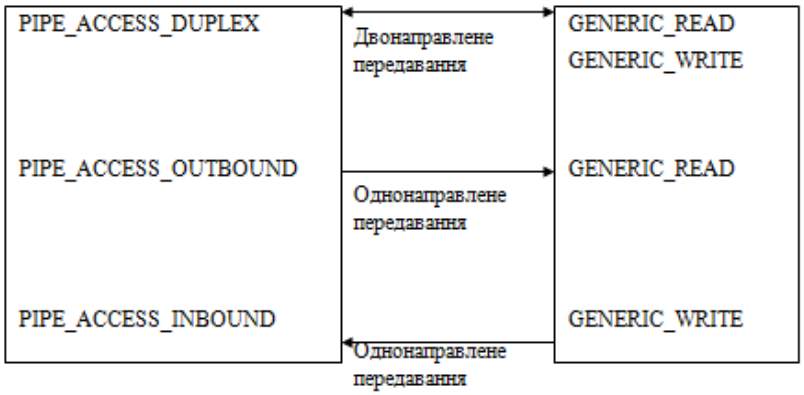


Рисунок 3.2 – Прапорці режимів і напрямків передавання

Прапорець FILE_FLAG_OVERLAPPED дозволяє функціям, що виконують операції читання, записування і з'єднання, негайно повертати значення, навіть якщо їх виконання вимагає істотних витрат часу.

Остання група прапорців – dwOpenMode (див. табл. 3.1) керує доступом сервера до дескриптора безпеки, створеного іменованим каналом. Їх використовують, якщо додатку потрібно відновити або змінити дескриптор після створення каналу. Прапорець WRITE_DACL дозволяє додатку оновлювати список обраного керування доступом DACL. Прапорець ACCESS_SYSTEM_SECURITY дозволяє отримати доступ до системного списку керування доступом SACL. Прапорець WRITE_OWNER дозволяє змінювати власника іменованого каналу і груповий ідентифікатор безпеки SID. Наприклад, можна заборонити доступ користувача до іменованого каналу, змінивши список DACL каналу за допомогою API-функцій.

dwPipeMode – визначає режими операцій читання, записування і очікування. При створенні каналу потрібно вказати по одному прапорцю з кожної категорії, об'єднавши їх за допомогою операції OR , як це показано у табл. 3.2.

Таблиця 3.2 – Прапорці режимів читання-записування іменованого каналу

Режим	Прапорець	Опис
Записування	PIPE_TYPE_BYTE	Дані записуються у канал потоком байтів
	PIPE_TYPE_MESSAGE	Дані записуються у канал потоком повідомлень
Читання	PIPE_READMODE_BYTE	Дані читаються з каналу потоком байтів
	PIPE_READMODE_MESSAGE	Дані читаються з каналу потоком
Очікування	PIPE_WAIT	Ввімкнено режим блокування
	PIPE_NOWAIT	Відключено режим блокування

Якщо при створенні побайтового каналу використовувалися прапорці PIPE_READMODE_BYTE | PIPE_TYPE_BYTE, то дані записуватимуться і читатимуться з каналу тільки потоком байтів. У цьому випадку не обов'язково врівноважувати кількість операцій читання і записування, оскільки дані не розділяються на окремі повідомлення. Наприклад, якщо у канал записано 500 байтів, одержувач може прочитувати по 100 байтів, до тих пір, доки не прочитає всі дані.

Щоб повідомлення мало чіткі межі, потрібно створити канал у режимі повідомлень, вказавши прапорці PIPE_READMODE_MESSAGE | PIPE_TYPE_MESSAGE. У цьому випадку необхідно врівноважувати кількість операцій читання і записування даних. Наприклад, якщо у канал записано 500 байтів, то для читання даних буде потрібно буфер розміром 500 байтів або більше, інакше функція ReadFile видасть помилку ERROR_MORE_DATA. Прапорець PIPE_TYPE_MESSAGE можна комбінувати з прапорцем PIPE_READMODE_BYTE. Це дозволить відправляти дані у режимі повідомлень, а при їх отриманні – зчитувати довільну кількість байтів. При такому способі передавання роздільники повідомлень ігноруються. Прапорець PIPE_TYPE_BYTE не можна використовувати з прапорцем PIPE_READMODE_MESSAGE, інакше функція CreateNamedPipe видасть помилку ERROR_INVALID_ PARAMETER, оскільки при такому способі записування потік даних не міститиме роздільників повідомлень. Прапорці PIPE_WAIT і PIPE_NOWAIT переводять канал у блокуючий і неблокуючий режими відповідно. Їх можна комбінувати з прапорцями режиму читання і

записування. У режимі блокування операції введення-виведення такі, як ReadFile, блокуються до тих пір, доки запит введення-виведення не буде виконаний повністю. Така поведінка є стандартною, якщо не вказано жодного прапорця. Якщо режим блокування відключено, операції введення-виведення повертають значення негайно. Проте даний режим не слід використовувати для досягнення асинхронного виконання операцій введення-виведення у додатках Win32. Асинхронне виконання функцій ReadFile і WriteFile досягається за допомогою перекритого введення-виведення.

nMaxInstances – визначає максимальну кількість екземплярів (описувачів) каналу, що одночасно може створити сервер. Екземпляр каналу – це з'єднання локального або видаленого клієнта з сервером іменованого каналу. Параметр може приймати значення від 1 до PIPE_UNLIMITED_INSTANCES. Наприклад, якщо сервер повинен одночасно підтримувати до п'яти з'єднань, параметру присвоюється значення 5. Якщо параметр дорівнює PIPE_UNLIMITED_INSTANCES, кількість екземплярів каналу обмежена тільки системними ресурсами.

nOutBufferSize і nInBufferSize – функції CreateNamedPipe визначають розміри вхідного і вихідного внутрішніх буферів. При створенні екземпляра каналу система кожного разу формує вхідний і (або) вихідний буфер, використовуючи резидентний пул (фізична пам'ять, що використовується операційною системою). Розмір буфера повинен бути не дуже великим, щоб система не вичерпала резидентний пул, і не дуже малим, щоб вмістити стандартні запити введення-виведення. При спробі записати дані більшого розміру система спробує автоматично розширити об'єм буфера, використовуючи резидентний пул. Оптимальні розміри – ті, які програма використовує при виклику функцій ReadFile і WriteFile.

nDefaultTimeOut – задає стандартний тайм-аут (час очікування з'єднання) у мілісекундах. Дія параметра розповсюджується тільки на ті клієнтські додатки, які визначають, чи можна встановити з'єднання з сервером за допомогою функції WaitNamedPipe.

lpSecurityAttributes – дозволяє додатку вказати дескриптор безпеки іменованого каналу і визначає, чи зможе дочірній процес успадковувати створений описувач. Якщо цей параметр дорівнює NULL, іменований канал використовує стандартний дескриптор безпеки, а описувач не може бути успадкований. Стандартний дескриптор безпеки означає, що

іменований канал має ті ж самі права доступу, що і процес, який його створив. Програма може надати доступ до каналу певним користувачам і групам, визначивши структуру SECURITY_DESCRIPTOR за допомогою API-функцій. Щоб відкрити доступ до сервера будь-яким клієнтам, у структурі SECURITY_DESCRIPTOR слід задати порожній список DACL.

Коли функція CreateNamedPipe поверне описувач іменованого каналу, сервер починає чекати з'єднання клієнтів. Для встановлення з'єднання використовується функція ***ConnectNamedPipe***, що визначена так:

```
BOOL ConnectNamedPipe(
HANDLE hNamedPipe,
LPOVERLAPPED lpOverlapped);
```

hNamedPipe – це описувач екземпляра іменованого каналу, повернутий функцією CreateNamedPipe. Якщо при створенні каналу використовувався прапорець FILE_FLAG_OVERLAPPED, параметр lpOverlapped дозволяє ConnectNamedPipe виконуватися асинхронно, тобто без блокування. Якщо цей параметр дорівнює NULL, ConnectNamedPipe блокується до тих пір, доки клієнт не встановить з'єднання з сервером.

Виклик функції ConnectNamedPipe завершується після встановлення з'єднання. Далі сервер відправляє і приймає дані за допомогою API-функцій WriteFile і ReadFile. Коли обмін даними з клієнтом завершено, сервер повинен закрити з'єднання, викликавши функцію DisconnectNamedPipe. У лістингу 3.1 показано, як написати сервер, здатний обмінюватися даними з одним клієнтом.

Лістинг 3.1 – Сервер іменованого каналу

```
//Server.cpp
#include <windows.h>
#include <stdio.h>
void main(void)
{
    HANDLE PipeHandle;
    DWORD BytesRead;
    CHAR buffer[256];
    if ((PipeHandle = CreateNamedPipe("\\\\.\\Pipe\\Jim",PIPE_
    ACCESS_DUPLEX, PIPE_TYPE_BYTE |PIPE_READMODE_BYTE,
    1, 0, 0, 1000, NULL)) == INVALID_HANDLE_VALUE)
```

```
    {
        printf("CreateNamedPipe failed with error %d\n",GetLastError());
        return;
    }
    printf("Server is now running\n");
    if(ConnectNamedPipe(PipeHandle, NULL)==0)
    {
        printf("ConnectNamedPipe failed with error %d\n",
        GetLastError());
        CloseHandle(PipeHandle);
        return;
    }
    if(ReadFile(PipeHandle, buffer,sizeof(buffer),&BytesRead,NULL)<=0)
    {
        printf("ReadFile failed with error %d\n",GetLastError());
        CloseHandle(PipeHandle);
        return;
    }
    buffer[BytesRead]=0;
    puts(buffer);
    if(DisconnectNamedPipe(PipeHandle)==0)
    {
        printf("DisconnectNamedPipe failed with error %d\n",
        GetLastError());
        CloseHandle(PipeHandle);
        return;
    }
    CloseHandle(PipeHandle);
}
```

3.8 Персоналізація

Іменовані канали дозволяють скористатися вбудованими можливостями захисту Windows для керування доступом клієнтів. Windows підтримує так звану персоналізацію, що дозволяє серверу виконуватися у контексті безпеки клієнта. Як правило, сервер іменованого каналу працює у контексті безпеки процесу, що його запустив. Наприклад,

якщо сервер іменованого каналу запущений користувачем з привілеями адміністратора, цей користувач одержує доступ практично до будь-яких ресурсів Windows. Дана ситуація небезпечна, адже структура SECURITY_DESCRIPTOR, передана у функцію CreateNamedPipe, відкриває доступ до каналу будь-яким користувачам. Встановивши з'єднання з клієнтом за допомогою функції ConnectNamedPipe, сервер може змусити свій потік виконуватися у контексті безпеки клієнта, викликавши API-функцію ***ImpersonateNamedPipeClient***:

BOOL ImpersonateNamedPipeClient(HANDLE hNamedPipe);

hNamedPipe – це описувач екземпляра іменованого каналу, повернутий функцією CreateNamedPipe. При виклику функції ImpersonateNamedPipeClient операційна система змінює контекст безпеки сервера на контекст безпеки клієнта. Тому, при звертанні до різних ресурсів (наприклад, файлів) сервер буде використовувати права доступу клієнта. Це дозволяє зберегти керування доступом до ресурсів, незалежно від того, хто запустив серверний додаток.

При роботі у контексті безпеки клієнта потік сервера використовує один з чотирьох основних рівнів персоналізації, що задаються клієнтом:

- анонімний (*Anonymous*);
- ідентифікація (*Identification*);
- персоналізація (*Impersonation*);
- делегування (*Delegation*).

Рівні персоналізації визначають ступінь, до якого сервер вправі представляти клієнта. Завершивши сеанс зв'язку, сервер повинен викликати функцію ***RevertToSelf***, щоб повернутися до початкового контексту безпеки. Ця функція не має параметрів:

BOOL RevertToSelf(VOID);

3.9 Деталі реалізації клієнта

Під реалізацією клієнта іменованого каналу розуміють розробку додатка, що може підключатися до сервера. Клієнти можуть тільки встановлювати з'єднання з існуючими на сервері каналами. Для створення простого клієнта потрібно виконати такі дії.

- Для перевірки наявності вільного екземпляра каналу викликати API-функцію WaitNamedPipe.

- Для встановлення з'єднання викликати API-функцію CreateFile.
- Для відправлення й одержання даних викликати API-функції WriteFile і ReadFile.
- Для завершення з'єднання викликати API-функцію CloseHandle.

Перед встановленням з'єднання клієнт повинен перевірити наявність на сервері вільного екземпляра іменованого каналу функцією ***WaitNamedPipe***:

BOOL WaitNamedPipe(
LPCTSTR lpNamedPipeName,
DWORD nTimeOut);

lpNamedPipeName – визначає назву іменованого каналу у форматі UNC.

nTimeOut – скільки часу клієнт буде чекати вільного екземпляра каналу.

У випадку успішного виконання функції WaitNamedPipe потрібно відкрити екземпляр іменованого каналу за допомогою API-функції ***CreateFile***:

HANDLE CreateFile(
LPCTSTR lpFileName,
DWORD dwDesiredAccess,
DWORD dwShareMode,
LPSECURITY_ATTRIBUTES lpSecurityAttibutes,
DWORD dwCreationDisposition,
DWORD dwFlagsAndAttributes,
HANDLE hTemplateFile);

lpFileName – назва каналу, що відкривається, у форматі UNC.

dwDesiredAccess – задає режим доступу і повинен дорівнювати GENERIC_READ при читанні чи GENERIC_WRITE – при записування даних у канал. Можна вказати обидва прапорці, об'єднавши їх за допомогою операції OR. Режим доступу повинен відповідати напрямку передавання (параметр dwOpenMode), заданому при створенні каналу. Наприклад, якщо канал створений із прапорцем PIPE_ACCESS_INBOUND, то клієнт вказує прапорець GENERIC_WRITE.

Параметр dwShareMode повинен дорівнювати нулю, оскільки у кожен момент часу тільки один клієнт може одержати доступ до екземпляра каналу. Параметр lpSecurityAttibutes повинен мати значення

NULL, якщо не потрібно, щоб дочірній процес успадковував описувач клієнта. Цей параметр не можна використовувати для керування доступом, тому що за допомогою функції CreateFile неможливо створити екземпляри іменованого каналу.

dwCreationDispcistion – варто визначити як OPEN_EXISTING, тоді функція CreateFile буде повертати помилку, якщо каналу не існує.

dwFlagsAndAttributes – обов'язково повинен містити прапорець FILE_ATTRIBUTE_NORMAL. Крім того, можна вказати прапорці FILE_FLAG_WRITE_THROUGH, FILE_FLAG_OVERLAPPED і SECURITY_SQOS_PRESENT, об'єднавши їх логічною операцією OR. Прапорці FILE_FLAG_WRITE_THROUGH і FILE_FLAG_OVERLAPPED використовуються для керування операціями введення-виведення. Прапорець SECURITY_SQOS_PRESENT визначає рівень персоналізації клієнта на сервері іменованого каналу. Рівні розгалуження задають ступінь свободи дій серверного процесу від імені клієнтського. Клієнт вказує цю інформацію при підключенні до сервера. Якщо клієнт задає прапорець SECURITY_SQOS_PRESENT, він також повинен вказати один чи кілька прапорців, перерахованих у наведеному далі списку.

- SECURITY_ANONYMOUS – задає рівень анонімності. Сервер не може одержати інформацію про клієнта і виконуватися у контексті безпеки клієнта.
- SECURITY_IDENTIFICATION – задає рівень ідентифікації. Сервер може одержати інформацію про клієнта, наприклад ідентифікатори і привілеї захисту, але не виконуватися у контексті безпеки клієнта. Це корисно, коли серверу необхідно ідентифікувати клієнта, але не потрібно грати його роль.
- SECURITY_IMPERSONATION – задає рівень персоналізації. Сервер може одержати інформацію про клієнта, а контекст безпеки клієнта поширюється тільки на локальну систему. Цей прапорець дозволяє серверу одержати доступ до будь-яких локальних ресурсів на сервері, так ніби він є клієнтом. Сервер не може представляти клієнта на віддалених системах.
- SECURITY_DELEGATION – задає рівень делегування. Сервер може одержати інформацію про клієнта і виконуватися у контексті безпеки клієнта на його локальній системі і віддалених системах.

Примітка. У Windows NT не реалізоване делегування безпеки, тому прапорець SECURITY_DELEGATION варто застосовувати, тільки якщо

сервер працює під керуванням Windows 2000.

• SECURITY_CONTEXT_TRACKING – задає динамічний режим спостерігання захисту. Якщо цей прапорець не зазначений, то режим статичний.

• SECURITY_EFFECTIVE_ONLY – вказує, що тільки включені аспекти контексту безпеки клієнта доступні серверу. Якщо цей прапорець не призначений, серверу доступні будь-які аспекти контексту безпеки клієнта.

Останній параметр функції CreateFile – hTemplateFile, не застосовується при роботі з іменованими каналами і повинен дорівнювати NULL. Після успішного виконання функції CreateFile клієнт може відправляти і приймати дані за допомогою функцій ReadFile і WriteFile. Завершивши передавання, клієнт закриває з'єднання функцією CloseHandle.

У лістингу 3.4 наведено код клієнта іменованого каналу. При успішному з'єднанні клієнт відправляє серверу повідомлення «This is a test».

Лістинг 3.4 – Клієнт іменованого каналу

```
// Client.cpp
#include <windows.h>
#include <stdio.h>
#define PIPE_NAME "\\\\.\\Pipe\\Jim"
void main(void)
{
    HANDLE PipeHandle;
    DWORD BytesWritten;
    if (WaitNamedPipe(PIPE_NAME, NMPWAIT_WAIT_FOREVER) == 0)
    {
        printf("WaitNamedPipe failed with error %d\n",GetLastError());
        return;
    }
    // Створення описувача файлу іменованого каналу
    if ((PipeHandle = CreateFile(PIPE_NAME,GENERIC_READ |
    GENERIC_WRITE, 0,(LPSECURITY_ATTRIBUTES) NULL,
    OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, (HANDLE) NULL))
```



```
    == INVALID_HANDLE_ VALUE)
    {
        printf("CreateFile failed with error %d\n", GetLastError());
        return;
    }
    if (WriteFile(PipeHandle, "This is a test", 14, &BytesWritten, NULL)
    == 0)
    {
        printf("WriteFile failed with error %d\n", GetLastError());
        CloseHandle(PipeHandle);
        return;
    }
    printf("Wrote %d bytes", BytesWritten);
    CloseHandle(PipeHandle);
}
```

3.10 Інші API-виклики

Ряд додаткових функцій полегшують роботу з іменованими каналами. Насамперед, розглянемо функції CallNamedPipe і TransactNamedPipe, що можуть істотно спростити код додатка. Обидві функції виконують операції читання і записування в одному виклику. Функція ***CallNamedPipe*** дозволяє клієнту підключитися до іменованого каналу, що працює у режимі повідомлень (і зачекати, доки не звільниться екземпляр каналу), записати і зчитати дані, а потім закрити канал. Фактично, вона є повноцінним клієнтом іменованого каналу:

```
BOOL CallNamedPipe(
LPCTSTR lpNamedPipeName,
LPVOID lpInBuffer,
DWORD nInBufferSize,
LPVOID lpOutBuffer,
DWORD nOutBufferSize,
LPDWORD lpBytesOfRead,
DWORD nTimeOut);
```

Параметр lpNamedPipeName визначає назву іменованого каналу у форматі UNC.

lpInBuffer і nInBufferSize – адреса і розмір буфера, який використовується додатком, для записування даних. Аналогічно, lpOutBuffer і nOutBufferSize визначають адресу і розмір буфера читання.

lpBytesOfRead – кількість байтів, зчитаних з каналу.

nTimeOut – час чекання звільнення каналу у мілісекундах.

Функція ***TransactNamedPipe*** використовується як у клієнтських, так і у серверних додатках. Вона поєднує операції читання і записування в одному API-виклику. Це оптимізує мережевий трафік за рахунок скорочення числа транзакцій, виконаних через перенаправлювач MSNP. Функція TransactNamedPipe визначена так:

BOOL TransactNamedPipe(
HANDLE hNamedPipe,
LPVOID lpInBuffer,
DWORD nInBufferSize,
LPVOID lpOutBuffer,
DWORD nOutBufferSize,
LPDWORD lpBytesRead,
LPOVERLAPPED lpOverlapped);

hNamedPipe – це описувач іменованого каналу, повернутий функцією CreateNamedPipe чи CreateFile.

lpInBuffer і nInBufferSize – визначають адресу і розмір буфера, використовуваного додатком для записування даних. Аналогічно, lpOutBuffer і nOutBufferSize позначають адресу і розмір буфера читання.

lpBytesRead – містить кількість байтів, зчитаних з каналу.

lpOverlapped – дозволяє функції TransactNamedPipe виконуватися асинхронно шляхом перекритого введення-виведення.

Група функцій: GetNamedPipeHandleState, SetNamedPipeHandleState GetNamedPipeInfo і PeekNamedPipe забезпечує більш гнучку взаємодію сервера і клієнта під час виконання. Наприклад, за допомогою цих функцій можна змінити режим роботи іменованого каналу з побайтового на режим повідомлень. Функція ***GetNamedPipeHandleState*** повертає інформацію про канал, включаючи режим роботи, кількість екземплярів каналу й інформацію про стан буферів. Інформація, що повертається цією функцією, може змінюватися у процесі роботи іменованого каналу. Функція визначена так

BOOL GetNamedPipeHandleState(

HANDLE hNamedPipe,
LPDWORD lpState,
LPDWORD lpCurInstances,
LPDWORD lpMaxCollectionCount,
LPDWORD lpCollectDataTimeout,
LPTSTR lpUserName,
DWORD nMaxUserNameSize);

hNamedPipe – це описувач іменованого каналу, повернутий функцією CreateNamedPipe чи CreateFile.

lpState – містить поточний режим роботи каналу і набуває значення PIPE_NOWAIT чи PIPE_READMODE_MESSAGE.

lpCurInstances – містить поточне число екземплярів каналу;

lpMaxCollectionCount – максимальна кількість байтів, що будуть накопичені на комп'ютері клієнта перед передаванням на сервер;

lpCollectDataTimeout – максимальний час у мілісекундах, що може пройти до того, як віддалений клієнт передасть інформацію з мережі;

lpUserName і nMaxUserNameSize – визначають буфер, що містить рядок з ім'ям користувача клієнтського додатка. Такий рядок повинен закінчуватися нулем.

Функція ***SetNamedPipeHandleState*** дозволяє змінити характеристики каналу, повернуті функцією GetNamedPipeHandleState:

BOOL SetNamedPipeHandleState(
HANDLE hNamedPipe,
LPDWORD lpMode,
LPDWORD lpMaxCollectionCount,
LPDWORD lpCollectDataTimeout);

hNamedPipe – це описувач іменованого каналу, повернутий функцією CreateNamedPipe чи CreateFile.

lpMode – задає режим роботи іменованого каналу.

lpMaxCollectionCount – містить максимальне число байтів, що будуть накопичені на комп'ютері клієнта перед передаванням на сервер.

lpCollectDataTimeout – визначає максимальний час у мілісекундах, що може пройти до того, як віддалений клієнт передасть інформацію з мережі.

Функція ***GetNamedPipeInfo*** повертає розмір буферів і максимальну кількість екземплярів каналу:

BOOL GetNamedPipeInfo(
HANDLE hNamedPipe,
LPDWORD lpFlags,
LPDWORD lpOutBufferSize,
LPDWORD lpInBufferSize,
LPDWORD lpMaxInstances);

hNamedPipe – описувач іменованого каналу, повернутий функцією CreateNamedPipe чи CreateFile.

lpFlags – вказує вид і режим роботи іменованого каналу і визначає, є він сервером чи клієнтом.

lpOutBufferSize і lpInBufferSize – містять розмір вихідного і вхідного внутрішніх буферів, lpMaxInstances – максимальна кількість екземплярів каналу.

Остання функція – PeekNamedPipe, дозволяє переглянути дані, що знаходяться у каналі, не стираючи їх із внутрішнього буфера. Наприклад, перевірити наявність вхідних даних і уникнути блокування функції ReadFile. Крім того, ця функція корисна, якщо необхідно перевірити дані перед одержанням: додаток може скорегувати розмір свого буфера залежно від розміру вхідного повідомлення. Функція ***PeekNamedPipe*** визначена так :

BOOL PeekNamedPipe (
HANDLE hNamedPipe,
LPVOID lpBuffer,
DWORD nBufferSize,
LPDWORD lpBytesRead,
LPDWORD lpTotalBytesAvial,
LPDWORD lpBytesLeftThisMessege);

hNamedPipe – описувач іменованого каналу, повернутий функцією CreateNamedPipe чи CreateFile.

lpBuffer і nBufferSize – визначають адресу і розмір приймаючого буфера.

1pBytesRead – містить кількість байтів, зчитаних з каналу у буфер lpBuffer, lpTotalBytesAvail – загальна кількість байтів, що можуть бути зчитані з каналу.

lpBytesLeftThisMessege – кількість байтів, що залишилося, у повідомленні, якщо канал працює у режимі повідомлень. Якщо

повідомлення не міститься у буфері lpBuffer, даний параметр містить кількість байтів, що залишилася. Якщо іменований канал працює у побайтовому режимі, параметр завжди містить нуль.

3.11 Контрольні питання

1. Що таке іменовані канали?
2. Які правила іменування каналів?
3. У яких режимах можуть працювати іменовані канали?
4. Які бібліотеки і файли заголовків потрібно підключити для роботи з іменованими каналами?
5. Як можна отримати інформацію про коди помилок при роботі іменованих каналів?
6. Опишіть інтерфейс функції CreateNamedPipe.
7. Які константи встановлюють напрямок передавання інформації при створенні іменованого каналу? Якому параметру присвоюються значення цих констант?
8. Які константи встановлюють режим читання і записування в іменований канал?
9. Якою змінною вказується, скільки каналів може одночасно створити сервер? Які значення може приймати дана змінна?
10. Яка функція дозволяє забезпечити роботу сервера у контексті безпеки клієнта? У який спосіб викликається дана функція?
11. Напишіть програму, що реалізує сервер іменованих каналів.
12. Напишіть програму клієнта іменованих каналів.
13. Які функції дозволяють отримувати та встановлювати параметри каналу? Опишіть інтерфейси цих функцій.
14. Яка функція дозволяє отримати кількість створених каналів та розміри їх буферів? Опишіть її інтерфейс.
15. Яка функція є повноцінним клієнтом іменованого каналу? Опишіть її інтерфейс.
16. Яка функція використовується для скорочення трафіка у серверах і клієнтах іменованих каналів? Опишіть її інтерфейс.
17. За допомогою якої функції можна переглянути дані, не витираючи буфер? Опишіть інтерфейс цієї функції.

4 ІНТЕРФЕЙС МЕРЕЖЕВОГО ПРОГРАМУВАННЯ NETBIOS

Інтерфейс Network Basic Input/Output System (NetBIOS) – стандартний інтерфейс прикладного програмування, розроблений Sytek Corporation для IBM у 1983 р. NetBIOS визначає програмний інтерфейс для мережевого зв'язку, але не обумовлює фізичний спосіб передавання даних мережею. У 1985 р. IBM створила протокол NetBIOS Extended User Interface (NetBEUI), інтегрований з інтерфейсом NetBIOS. Програмний інтерфейс NetBIOS незабаром набув такої популярності, що постачальники програмного забезпечення почали реалізовувати його для інших протоколів, таких як TCP/IP і IPX/SPX.

4.1 Мережева модель OSI

Модель відкритого з'єднання систем (*Open Systems Interconnect, OSI*) забезпечує високорівневе подання мережевих систем. Сім її рівнів цілком описують фундаментальні мережеві концепції: від додатка до способу фізичного передавання даних. Ось ці рівні:

- прикладний – надає інтерфейс користувача для передавання даних між програмами;
- представницький – форматує дані;
- сеансовий – керує зв'язком між двома вузлами;
- транспортний – забезпечує передавання даних (надійне або ненадійне);
- мережевий – підтримує механізм адресації між вузлами і маршрутизацію пакетів даних;
- канальний – керує взаємодією між вузлами на фізичному рівні, відповідає за групування даних, переданих через фізичний носій;
- фізичний – фізичний носій, відповідальний за передавання даних у вигляді електричних сигналів.

У цій моделі NetBIOS відноситься до сеансового і транспортного рівнів.

4.2 Особливості Microsoft NetBIOS

Як уже згадувалося, існують реалізації NetBIOS для різних

мережевих протоколів, що дозволяє використовувати протоколи TCP/IP, NetBEUI чи навіть IPX/SPX. Це дозволяє коректно написаному додатку NetBIOS виконуватися майже на будь-якому комп'ютері, незалежно від фізичної мережі. Однак є кілька нюансів. Для того, щоб два додатки NetBIOS могли зв'язатися один з одним через мережу, вони повинні виконуватися на робочих станціях, що мають принаймні один спільний транспортний протокол. Наприклад, якщо на одному комп'ютері встановлений тільки TCP/IP, а на іншому комп'ютері – тільки NetBEUI, то прикладні програми NetBIOS на першому комп'ютері не зможуть зв'язатися з прикладними програмами на другому комп'ютері.

Крім того, тільки деякі протоколи реалізують інтерфейс NetBIOS. Наприклад, Microsoft TCP/IP і NetBEUI роблять це за замовчуванням, а IPX/SPX – ні. Тому Microsoft пропонує версію IPX/SPX, що реалізує цей інтерфейс. Як правило, під час встановлення протоколів видно, чи підтримує версія протоколу IPX/SPX можливості NetBIOS. Наприклад, разом з Windows 2000 постачається NWLink IPX/SPX/NetBIOS Compatible Transport Protocol. Те, що цей протокол підтримує NetBIOS, прямо випливає з його назви. У Windows 98 у вікні властивостей протоколу IPX/SPX є прапорець, за допомогою якого можна встановити підтримку NetBIOS для IPX/SPX.

Важливо, що NetBEUI – немаршрутизований протокол. Якщо між клієнтом і сервером є маршрутизатор, прикладні програми на цих комп'ютерах не зможуть зв'язатися. TCP/IP і IPX/SPX – маршрутизовані протоколи і не мають такого обмеження. Отже, для використання NetBIOS, бажано встановити у мережі принаймні один з маршрутизованих транспортних протоколів.

4.3 Номери LANA

Ключ до розуміння NetBIOS – номери мережевих адаптерів (*Local area network adapter number, LANA number*). У початкових реалізаціях NetBIOS кожному фізичному мережевому адаптеру присвоювалось унікальне значення – номер мережевої плати. У Win32 це стало проблематичним, тому що робоча станція може мати і декілька мережевих протоколів, і декілька плат мережевого інтерфейсу.

Номер LANA відповідає унікальній комбінації мережевого адаптера

і транспортного протоколу. Так, якщо робоча станція має дві мережевих плати і два підтримуючих NetBIOS транспорти (наприклад, TCP/IP і NetBEUI), буде присвоєно чотири номери LANA. Номери можуть відповідати комбінаціям адаптера з протоколом приблизно у такий спосіб:

0 –«TCP/IP – мережевий адаптер 1»;

1 –«NetBEUI – мережевий адаптер 1»;

2 –«TCP/IP – мережевий адаптер 2»;

3 –«NetBEUI – мережевий адаптер 2».

Номери LANA лежать у діапазоні від 0 до 9, і операційна система призначає їх у випадковому порядку. Крім LANA 0, що має особливий сенс – це номер "за замовчуванням". Коли з'явився інтерфейс NetBIOS, більшість операційних систем підтримувало єдиний номер LANA і багато додатків були жорстко запрограмовані на роботу тільки з LANA 0. Для зворотної сумісності можна напряму призначити LANA 0 конкретному протоколу.

При розробці додатка NetBIOS завжди пишеться код, що може обробляти з'єднання на будь-якому номері LANA. Припустимо, на комп'ютері А серверний додаток NetBIOS слухає клієнтів на LANA 2. На комп'ютері А LANA 2 відповідає протоколу TCP/IP. На комп'ютері Б клієнтський додаток зв'язується через LANA 2 з комп'ютером А. Однак LANA 2 на комп'ютері Б відповідає NetBEUI. Додатки не зможуть зв'язатися один з одним, хоча їм доступні протоколи TCP/IP і NetBEUI. Щоб усунути цю невідповідність, серверний додаток на комп'ютері А повинен слухати клієнтські з'єднання на кожному доступному номері LANA. Аналогічно, клієнтський додаток на комп'ютері Б повинен намагатися зв'язатися на кожному номері LANA. Звичайно, наявність коду, який може обробляти з'єднання на будь-якому номері LANA, не означає, що цей код буде працювати, якщо у двох комп'ютерів не знайдеться жодного спільного протоколу.

4.4 Імена NetBIOS

Тепер перейдемо до імен NetBIOS. Процес реєструє своє ім'я на кожному номері LANA, з яким йому потрібно зв'язатися. Ім'я NetBIOS має довжину 16 символів (16-й символ зарезервовано для спеціального призначення). При доданні імені у таблицю імен, потрібно очистити буфер

імен. У середовищі Win32 кожен процес має таблицю імен NetBIOS для кожного доступного номера LANA. Додавання імені для LANA 0 означає, що додаток доступний тільки клієнтам, що з'єднуються на цьому LANA 0. Максимально до кожного номера LANA можуть бути додані 254 імені, вони пронумеровані від 1 до 254 (0 і 255 зарезервовані для системи). Утім, кожна ОС задає максимальний номер за замовчуванням, менший 254. Його можна змінити при перевизначенні кожного номера LANA.

Є два типи імен NetBIOS: унікальне і групове. Якщо деякий процес у мережі спробує зареєструвати вже наявне унікальне ім'я, буде видана помилка дублювання імені. Як відомо, імена комп'ютерів у мережах Microsoft – імена NetBIOS. Коли комп'ютер завантажується, він реєструє своє ім'я на локальному сервері Windows Internet Naming Server (WINS), що повідомляє про помилку, якщо інший комп'ютер уже використовує те ж ім'я. Сервер WINS підтримує список усіх зареєстрованих імен NetBIOS. Разом з іменем можуть зберігатися і відомості про протокол. Наприклад, у мережах TCP/IP WINS запам'ятовує IP-адресу комп'ютера, що зареєстрував ім'я NetBIOS. Якщо мережа не має сервера WINS, комп'ютер шляхом широкомовної розсилки повідомлення перевіряє, чи немає у мережі такого ж імені. Якщо ніякий інший комп'ютер не заперечує повідомлення, мережа дозволяє відправнику використовувати заявлене ім'я.

Групові імена використовуються, щоб відправляти чи, навпаки, одержувати дані, призначені для багатьох одержувачів. Ім'я групи не обов'язково повинно бути унікальним. Групові імена використовуються для багатоадресного розсилання.

В іменах NetBIOS 16-й символ визначає більшість мережевих служб Microsoft, імена служб і груп для WINS-сумісних комп'ютерів. WINS реєструє сервер напряму, а для реєстрації імен інших комп'ютерів застосовуються широкомовні повідомлення у локальній підмережі. Для того, щоб одержати інформацію про зареєстровані на локальному (чи віддаленому) комп'ютері імена NetBIOS, потрібно використати утиліту Nbtstat. У табл. 4.1 наведено приклад даних про зареєстровані імена NetBIOS, що команда Nbtstat-n видала для користувача Davemac, який увійшов в основний контролер домену і працює під керуванням Windows NT Server разом з Internet Information Server.

Утиліта Nbtstat встановлюється разом із протоколом TCP/IP. Вона може також опитувати таблиці імен на віддалених комп'ютерах,

використовуючи параметр *-a* з іменем віддаленого комп'ютера чи параметр *-A* з його IP-адресою.

Таблиця 4.1 – Приклад таблиці імен NetBIOS

Ім'я	16-й байт	Тип імені	Служба
DAVEMAC 1	<00>	Унікальне	Ім'я служби робочої станції
DAVEMAC 1	<20>	Унікальне	Ім'я служби сервера
DAVEMACD	<00>	Групове	Ім'я домену
DAVEMACD	<1C>	Групове	Ім'я контролера домену
DAVEMACD	<1B>	Унікальне	Ім'я координатора мережі
DAVEMAC1	<03>	Унікальне	Ім'я відправника
Inet~Services	<1C>	Групове	Групове ім'я Internet Information Server
IS~DAVEMAC1	<00>	Унікальне	Унікальне ім'я Internet Information Server
DAVEMACl +++++++	<BF>	Унікальне	Унікальне ім'я мережевого монітора

Опишемо стандартні значення 16-го байта, що додаються у кінець унікальних NetBIOS-імен комп'ютерів мережевими службами Microsoft:

<00> – ім'я служби робочої станції (NetBIOS-ім'я комп'ютера);

<03> – ім'я служби повідомлень, що використовується при одержанні й відправленні повідомлень. Зареєстровано сервером WINS для служби повідомлень на клієнті WINS і звичайно додається у кінець імені комп'ютера і користувача, що увійшов у систему;

<1B> – ім'я координатора мережі (master browser) домену. Визначає основний контролер домену і вказує, яких клієнтів і інших оглядачів використовувати для контакту з координатором мережі домену;

<06> – серверна служба віддаленого доступу (RAS);

<1F> – служба мережевого динамічного обміну даними (Network Dynamic Data Exchange, NetDDE);

<20> – ім'я служби на сервері, що використовується для надання точок підключення до загальних файлів;

<21> – клієнт RAS;

<BE> – агент мережевого монітора;

<BF> – утиліта Network Monitor (Мережевий монітор).

А тепер опишемо задані за замовчуванням символи 16-го байта, що додаються у кінець звичайно використовуваних групових імен NetBIOS.

<1C> – групове ім'я домену, що містить список певних адрес комп'ютерів, які його зареєстрували. Контролер домену це ім'я реєструє, WINS обробляє його як доменну групу, а кожен член групи повинен індивідуально оновити своє ім'я або буде виключений. Доменна група обмежена 25 іменами. Якщо у результаті реплікації статичне 1С-ім'я конфліктує з динамічним 1С-ім'ям на іншому сервері WINS, для відповідних учасників додається статичний комбінований запис;

<1D> – ім'я координатора мережі (master browser), що використовується клієнтами для звертання до нього. У підмережі може бути лише один координатор. Сервери WINS повертають позитивну відповідь на реєстрацію імені домену, але не зберігають це ім'я у своїх базах даних. Якщо комп'ютер надсилає запит на ім'я домену сервера WINS, той повертає негативну відповідь. Якщо комп'ютер, що зробив запит, являє собою h- чи m-вузол, він потім виконує широкомовне розсилання запиту імені, щоб дозволити його. Тип вузла визначає спосіб дозволу імені клієнтом. Клієнти, що знаходяться у режимі b-вузла, виконують широкомовне розсилання пакетів для оповіщення про себе і дозволу імен NetBIOS. При дозволі у режимі p-вузла зв'язок із сервером WINS здійснюється у стилі "точка – точка". При дозволі імен у режимі m-вузла (змішаному) спочатку встановлюється режим b-вузла, а потім при необхідності – режим p-вузла. Останній метод дозволу – h-вузол (гібридний). У цьому режимі спочатку застосовується реєстрація і дозвіл у режимі p-вузла, а у випадку невдачі – у режимі b-вузла. За замовчуванням у Windows використовується режим h-вузла;

<1E> – звичайне групове ім'я. Оглядачі можуть виконувати широкомовне розсилання для даного імені і слухати на ньому, щоб вибрати координатора мережі. Ці широкомовні розсилання ефективні лише у локальній підмережі і не ретранслюються маршрутизаторами;

<20> – ім'я Інтернет-групи. Реєструється серверами WINS, щоб ідентифікувати групи комп'ютерів з адміністративною метою. Наприклад, можна зареєструвати групове ім'я «printers» для ідентифікації адміністративної групи серверів друку.

MSBROWSE – додається у кінець імені домену замість 16-го символа. Це ім'я широкомовно розсилається локальною підмережею, щоб повідомити про домен іншим її координаторам.

Така кількість специфікаторів може здаватися надмірною: швидше

за все, не буде необхідності використовувати їх у іменах NetBIOS, але все-одно, потрібно мати їх на увазі. Щоб уникнути випадкових колізій між іменами NetBIOS не можна використовувати специфікатори унікальних імен. Ще більш обережно потрібно ставитись до групових імен: якщо ім'я буде суперечити існуючому груповому імені, помилка видана не буде, і програма почне одержувати дані, призначені для когось іншого.

4.5 Особливості NetBIOS

NetBIOS пропонує як служби, що потребують логічного з'єднання, так і служби без встановлення з'єднання, тобто дейтаграмні (*datagram*). Перші дозволяють двом об'єктам встановлювати сеанс або віртуальний канал між ними. Сеанс – це двосторонній комунікаційний потік, через який об'єкти обмінюються повідомленнями. Служби, що потребують з'єднання, гарантують доставку будь-яких даних між двома кінцевими комп'ютерами. Сервер звичайно реєструє себе під певним відомим ім'ям, і клієнти шукають це ім'я, щоб зв'язатися із сервером. Стосовно NetBIOS, процес сервера додає його ім'я у таблицю імен для кожного номера LANA, з яким потрібен зв'язок. Клієнти на інших комп'ютерах перетворюють ім'я служби у ім'я комп'ютера і потім запитують з'єднання у серверного процесу. Отже, для встановлення такого зв'язку необхідно спочатку виконати зазначені дії, тому ініціалізація з'єднання пов'язана з деякими витратами. При встановленні сеансу гарантується доставка пакетів і їхній порядок, хоча зв'язок і заснований на повідомленнях. Тобто, якщо клієнт, що підключається, направить запит на читання, сервер поверне тільки один пакет даних у потоці, навіть якщо клієнт забезпечив буфер для приймання декількох пакетів.

При використанні служб, які не вимагають логічного з'єднання (дейтаграмних), сервер реєструється під конкретним ім'ям, а клієнт просто посилає дані у мережу, не встановлюючи заздалегідь ніякого з'єднання. Клієнт направляє дані на NetBIOS-ім'я серверного процесу. Дейтаграмні служби ефективніші, оскільки не витрачають час і ресурси на встановлення з'єднання. З іншого боку, дейтаграмні служби не дають гарантій доставки і порядку повідомлень. Наприклад, клієнт відправляє тисячі байтів даних на сервер, що два дні назад був зупинений через відмову. Відправник ніколи не одержить ніяких повідомлень про помилку,

якщо тільки він не повинен отримувати відповідь від сервера (якщо протягом досить тривалого періоду часу немає відповіді, то можна зрозуміти, що щось не так).

4.6 Основи програмування NetBIOS

Тепер розглянемо API-інтерфейс NetBIOS. Він є елементарним, оскільки містить тільки одну функцію:

UCHAR Netbios(PNCB pNCB);

Всі оголошення функцій, константи та інші параметри для NetBIOS визначені у заголовному файлі. Єдина бібліотека, необхідна для компонування додатків NetBIOS – Netapi32.lib. Найбільш важлива особливість цієї функції – параметр pNCB, що є вказівником на блок мережевого керування (*network control block, NCB*). Даний параметр є вказівником на структуру NCB, що містить всю інформацію, необхідну функції NetBIOS для виконання команди NetBIOS. Визначається дана структура так:

```
typedef struct _NCB
{
    UCHAR    ncb_command;
    UCHAR    ncb_retcode;
    UCHAR    ncb_lsn;
    UCHAR    ncb_num;
    PUCHAR   ncb_buffer;
    WORD     ncb_length;
    UCHAR    ncb_callname[NCBNAMSZ];
    UCHAR    ncb_name[NCBNAMSZ];
    UCHAR    ncb_rto;
    UCHAR    ncb_sto;
    void     (*ncb_post) (struct _NCB *);
    UCHAR    ncb_lana_num;
    UCHAR    ncb_cmd_cplt;
    UCHAR    ncb_reserve[10];
    HANDLE   ncb_event;
} * PNCB, NCB;
```

Не всі члени структури будуть використовуватися у кожному

виклику NetBIOS. Крім того, деякі з полів даних є вихідними параметрами (іншими словами, задаються після повернення з виклику NetBIOS). Слід відзначити, що завжди потрібно спочатку встановлювати у нульове значення всі поля структури NCB, а потім задавати значення у тих полях, де це необхідно. Далі описано призначення кожного поля.

ncb_command – вказує виконувану команду NetBIOS. Багато команд можуть виконуватися синхронно або асинхронно порозрядним логічним додаванням прапорця ASYNCH (0x80) і команди.

ncb_retcode – визначає код повернення для даної операції. Функція присвоює цьому полю значення NRC_PENDING, коли відбувається асинхронна операція.

ncb_lsn – визначає номер локального сеансу, унікально ідентифікуючого його у поточному оточенні. Функція повертає новий номер сеансу після успішної команди NCBCALL або NCBLISTEN.

ncb_num – вказує номер локального мережевого імені. Новий номер повертається для кожного виклику команди NCBADDNAME чи NCBADDGRNAME. Потрібно використовувати коректний номер з усіма дейтаграмними командами.

ncb_buffer – вказує на буфер даних. Для команд, що відправляють дані, цей буфер містить дані, які відправляються, а для команд, що одержують дані, – дані, які повертаються функцією NetBIOS. Для інших команд, типу NCBENUM, буфер буде визначений структурою LANA_ENUM.

ncb_length – вказує довжину буфера у байтах. Для команд приймання NetBIOS присвоює цьому полю значення, яке дорівнює кількості отриманих байтів. Якщо буфер недостатньо великий, NetBIOS повертає помилку NRC_BUFLEN.

ncb_callname – вказує ім'я віддаленого додатка.

ncb_name – вказує ім'я, під яким відомий локальний додаток.

ncb_rto – задає час очікування (тайм-аут) для операцій приймання. Значення визначене у 500-мілісекундних одиницях (нуль – означає нульовий час чекання) і задано для команд NCBCALL і NCBLISTEN, що впливає на команди NCBRECV.

ncb_sto – аналогічно задає час очікування для операцій відправлення. Це впливає на команди NCBSEND і NCBCHAINSEND.

ncb_post – вказує адресу процедури, яку потрібно викликати після

завершення асинхронної команди. Функція визначена як void CALLBACK PostRoutine (PNCB pncb), де pncb вказує на блок мережевого керування завершеної команди.

ncb_lana_num – вказує номер LANA для виконання команди.

ncb_cmd_cplt – визначає код повернення для операції. NetBIOS присвоює цьому полю значення NRC_PENDING, коли відбувається асинхронна операція.

ncb_reserve – зарезервовано і повинно дорівнювати нулю.

ncb_event – вказує опис об'єкта події Windows у вільному стані (*nonsignaled state*). Коли асинхронна команда завершується, подія переходить у зайнятий стан (*signaled state*). Потрібно використовувати тільки ручне скидання подій. Це поле дорівнює нулю, якщо у полі ncb_command не заданий прапорець ASYNCH чи якщо значення нуля ncb_post відмінне від нуля. Інакше NetBIOS повертає помилку NRC_ILLCMD.

4.7 Команди NetBIOS

Всього є двадцять чотири команди NetBIOS. Кожна з команд NetBIOS для свого виконання потребує задання параметрів структури NCB. Результат виконання команди повертається функцією Netbios. Крім того, функція Netbios може записувати результати виконання команди у деякі поля структури NCB. Існують такі команди NetBIOS:

1. NCBADDGRNAME – додає групове ім'я у таблицю імен на заданому номері LANA;
2. NCBADDNAME – додає унікальне ім'я у таблицю імен на заданому номері LANA;
3. NCBASTAT – повертає дані про стан віддаленого або локального мережевого адаптера;
4. NCBCALL – встановлює мережеве з'єднання з іншим комп'ютером;
5. NCBCANCEL – скасовує попередню команду;
6. NCBCHAINSEND – відправляє вміст двох буферів віддаленому отримувачу;
7. NCBCHAINSENDNA – відправляє вміст двох буферів віддаленому отримувачу, не чекаючи підтвердження;

8. NCBDELNAME – видаляє ім'я з таблиці імен на вказаному номері LANA;

9. NCBDGRECV – отримує дейтаграму від одного комп'ютера;

10. NCBDGRECVBC – отримує широкомовну дейтаграму;

11. NCBDGSEND – відправляє дейтаграму одному комп'ютеру;

12. NCBDGSENDBC – широкомовно відправляє дейтаграму;

13. NCBENUM – задає номери LANA;

14. NCBFINDNAME – знаходить комп'ютер у мережі;

15. NCBHANGUP – аварійно завершує сеанс зв'язку;

16. NCBLANSTALERT – повідомляє про збій у системі, який продовжувався протягом більше хвилини;

17. NCBLISTEN – прослуховує запити на з'єднання від інших процесів;

18. NCBRECV – отримує дані від комп'ютера, з яким встановлено сеанс зв'язку;

19. NCBRECVANY – отримує дані від будь-якого сеансу;

20. NCBRESET – скидає вказаний номер LANA;

21. NCBSEND – відправляє дані комп'ютеру, з яким встановлено сеанс зв'язку;

22. NCBSENDNA – відправляє дані комп'ютеру, з яким встановлено сеанс зв'язку, не чекаючи підтвердження;

23. NCBSTAT – отримує дані про стан сеансу;

24. NCBUNLINK – відміняє прив'язку до адаптера.

У таблиці 4.2 вказано, як задавати поля структури NCB для кожної з цих команд.

4.8 Синхронний і асинхронний виклик

Функцію NetBIOS можна викликати у синхронному або асинхронному режимах. Однак, усі команди, які викликає функція NetBIOS, виконуються у синхронному режимі. Синхронний режим виконання функції NetBIOS означає, що її робота блокується до того часу, доки викликана нею команда не завершить свою роботу. Наприклад, при синхронному виклику команди NCBLISTEN запит до NetBIOS не повертається до того часу, доки клієнт не встановить з'єднання чи не виникне помилка.

Таблиця 4.2 – Задання параметрів структури NCB для команд NetBIOS

Команди	Параметри	ncb_command	ncb_retcode	ncb_lsn	ncb_num	ncb_buffer	ncb_length	ncb_callname	ncb_name	ncb_rto	ncb_sto	ncb_post	ncb_lana_num	ncb_cmd_cplt	ncb_event
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
NCBADDGRNAME	I/O	I	O	-	O	-	-	-	I	-	-	I	I	O	I
	Задан.	+	-	-	-	-	-	-	+	-	-	-	+	-	-
NCBADDNAME	I/O	I	O	-	O	-	-	-	I	-	-	I	I	O	I
	Задан.	+	-	-	-	-	-	-	+	-	-	-	+	-	-
NCBASTAT	I/O	I	O	-	-	IO	IO	I	-	-	-	I	I	O	I
	Задан.	+	-	-	-	+	+	+	-	-	-	-	+	-	-
NCBCALL	I/O	I	O	O	-	-	-	I	I	I	I	I	I	O	I
	Задан.	+	-	-	-	-	-	+	+	-	-	-	+	-	-
NCBCANCEL	I/O	I	O	-	-	I	-	-	-	-	-	-	I	O	I
	Задан.	+	-	-	-	+	-	-	-	-	-	-	+	-	-
NCBCHAINSEND	I/O	I	O	I	-	I	I	I	-	-	-	I	I	O	I
	Задан.	+	-	+	-	+	+	+	-	-	-	-	+	-	-
NCBCHAINSENDNA	I/O	I	O	I	-	I	I	I	-	-	-	I	I	O	I
	Задан.	+	-	+	-	+	+	+	-	-	-	-	+	-	-
NCBDELNAME	I/O	I	O	-	-	-	-	-	I	-	-	I	I	O	I
	Задан.	+	-	-	-	-	-	-	+	-	-	-	+	-	-
NCBDGRECV	I/O	I	O	-	I	O	IO	O	-	-	-	I	I	O	I
	Задан.	+	-	-	+	+	+	-	-	-	-	-	+	-	-
NCBDGRECVBC	I/O	I	O	-	I	I	IO	O	-	-	-	I	I	O	I
	Задан.	+	-	-	+	+	+	-	-	-	-	-	+	-	-
NCBDGSEND	I/O	I	O	-	I	I	I	I	-	-	-	I	I	O	I
	Задан.	+	-	-	+	+	+	+	-	-	-	-	+	-	-
NCBDGSENDBC	I/O	I	O	I	I	I	-	-	-	-	-	I	I	O	I
	Задан.	+	-	+	+	+	-	-	-	-	-	-	+	-	-
NCBENUM	I/O	I	O	-	-	I	I	-	-	-	-	-	I	O	I
	Задан.	+	-	-	-	+	+	-	-	-	-	-	+	-	-

Продовження таблиці 4.2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
NCBFINDNAME	I/O	I	O	-	-	IO	I	I	-	-	-	I	I	O	I
	Задан.	+	-	-	-	+	+	+	-	-	-	-	+	-	-
NCBHANGUP	I/O	I	O	I	-	-	-	-	-	-	-	I	I	O	I
	Задан.	+	-	+	-	-	-	-	-	-	-	-	+	-	-
NCBLANSTALERT	I/O	I	O	-	-	-	-	-	-	-	-	-	-	O	I
	Задан.	+	-	-	-	-	-	-	-	-	-	-	-	-	-
NCBLISTEN	I/O	I	O	O	-	-	-	IO	I	I	I	I	I	O	I
	Задан.	+	-	-	-	-	-	+	+	-	-	-	+	-	-
NCBRECV	I/O	I	O	I	I	I	IO	-	-	-	-	I	I	O	I
	Задан.	+	-	+	+	+	+	-	-	-	-	-	+	-	-
NCBRECVANY	I/O	I	O	O	IO	I	IO	-	-	-	-	I	I	O	I
	Задан.	+	-	-	+	+	+	-	-	-	-	-	+	-	-
NCBRESET	I/O	I	O	I	I	-	-	-	-	-	-	-	I	O	I
	Задан.	+	-	+	+	-	-	-	-	-	-	-	+	-	-
NCBSEND	I/O	I	O	I	-	I	I	-	-	-	-	I	I	O	I
	Задан.	+	-	+	-	+	+	-	-	-	-	-	+	-	-
NCBSENDNA	I/O	I	O	I	-	I	I	-	-	-	-	I	I	O	I
	Задан.	+	-	+	-	+	+	-	-	-	-	-	+	-	-
NCBSTAT	I/O	I	O	-	O	I	I	-	I	-	-	I	I	O	I
	Задан.	+	-	-	-	+	+	-	+	-	-	-	+	-	-

Для асинхронного виклику функції потрібно виконати логічну операцію OR над командою NetBIOS і прапорцем ASYNCH. При використанні прапорця ASYNCH необхідно у полі ncb_post визначити процедуру, яка буде викликатись після виконання команди або визначити опис події у полі ncb_event. Після асинхронного виклику функція NetBIOS задає полю ncb_cmd_cplt структури NCB значення NRC_PENDING і одразу ж завершується, повертаючи значення NRC_GOODRET (0x00). Після завершення команди полю ncb_cmd_cplt присвоюється значення, повернуте командою. Після свого завершення команда NetBIOS також присвоює полю ncb_retcode свій код повернення.

4.9 Приклад додатка NetBIOS

Нижче наведено лістинг додатка NetBIOS, що визначає MAC-адресу мережевої плати.

```
#include "stdafx.h"
#include "windows.h"
#include "NB30.h"
#include "stdio.h"
#include "conio.h"
#include "mbstring.h"
struct Adapter
{
        ADAPTER_STATUS adapt;
        NAME_BUFFER NameBuff[30];
};
int main(int argc, char* argv[])
{
        NCB n;
        memset(&n,0,sizeof(n));
        n.ncb_command=NCBRESET;
        n.ncb_lana_num=0;
        UCHAR uRetCod;
        uRetCod=Netbios(&n);
        if(uRetCod!=0)
        {
              puts("Error!");
              return 0;
        }
        n.ncb_command=NCBASTAT;
        n.ncb_lana_num=0;
        _mbscpy(n.ncb_callname,(UCHAR*)"* ");
        Adapter al;
        n.ncb_buffer=(UCHAR*)&al;
        n.ncb_length=sizeof(al);
        uRetCod=Netbios((NCB*)&n);
        if(uRetCod==0)
```

```
    {
        printf("The Ethernet Number is: %02X-%02X-%02X-%02X-%02
        X-%02X\n", al.adapt.adapter_address[0],
        al.adapt.adapter_address[1], al.adapt.adapter_address[2],
        al.adapt.adapter_address[3], al.adapt.adapter_address[4],
        al.adapt.adapter_address[5]);
    }
    getch();
    return 0;
}
```

4.10 Контрольні питання

1. Назвіть рівні системи мережевих протоколів OSI і опишіть призначення кожного рівня.
2. Які обмеження накладає інтерфейс програмування NetBIOS на транспортні протоколи? Наведіть приклади.
3. Що таке номери LANA? Як можна їх задати? Як використовувати номери LANA у додатках? Наведіть приклади.
4. Опишіть призначення, формат імен і види унікальних імен інтерфейсу програмування NetBIOS.
5. Опишіть призначення, формат імен і види групових імен інтерфейсу програмування NetBIOS.
6. Опишіть послідовність встановлення зв'язку між віддаленими процесами за допомогою NetBIOS.
7. Яка функція використовується у NetBIOS? Який вона має інтерфейс?
8. Опишіть призначення параметрів функції Netbios.
9. Для чого використовується структура ncb? Опишіть поля даної структури.
10. Як виконується синхронний і асинхронний виклик функції Netbios? Наведіть приклади.
11. Напишіть приклад програми з використанням інтерфейсу програмування NetBIOS.
12. Скільки існує команд NetBIOS? Назвіть деякі з них. Як виконати команду NetBIOS асинхронно?

13. Які протоколи підтримують NetBIOS?
14. Як використовується поле ncb_post структури ncb при виконанні асинхронних команд?
15. Які поля структури ncb використовуються для встановлення часу очікування операцій приймання і відправлення інформації. У яких одиницях задається цей час.
16. Чи підтримує інтерфейс програмування NetBIOS встановлення сеансу зв'язку?
17. У чому особливість номера LANA, що дорівнює нулю? Яка максимальна кількість номерів LANA може бути в операційній системі?
18. Яка команда задає номери LANA? Опишіть, як задаються параметри функції Netbios для виконання даної команди.
19. Що означає 16-й байт у іменах NetBIOS?
20. Яка різниця між індивідуальними і груповими іменами інтерфейсу мережевого програмування NetBIOS? Де ще, крім додатків, використовуються імена NetBIOS?
21. Які заголовкові файли та які бібліотеки потрібно підключити у додатках для використання NetBIOS?
22. Як резервуються імена комп'ютерів у мережах, що не мають сервера WINS?
23. Скільки комп'ютерів можуть входити до доменної групи?
24. Які недоліки має протокол NetBEUI?
25. Як можна отримати список зареєстрованих на комп'ютері імен NetBIOS?
26. Яким чином сервер WINS реєструє імена комп'ютерів у локальній мережі?
27. Які значення повертають функція Netbios і команда при асинхронному виконанні?
28. Які дії потрібно виконати перед встановленням полів структури ncb?
29. Як використовується поле ncb_cmd_cplt структури ncb при виконанні асинхронних команд?

5 ІНТЕРФЕЙС МЕРЕЖЕВОГО ПРОГРАМУВАННЯ WINSOCK

5.1 Характеристики протоколів

Winsock – це сучасна бібліотека, призначена для створення інтерфейсу, незалежного від транспортного протоколу. Вона реалізована версіями Winsock 1.1 і Winsock 2.2. Програма, написана для Winsock 1.1, зможе виконуватись і з бібліотекою Winsock 2.2. Далі описані характеристики транспортних протоколів, що використовує Winsock.

5.1.1 Режими передавання

Транспортні протоколи можуть бути орієнтованими на передавання повідомлень (дейтаграм) або на передавання потоків.

Протокол називають орієнтованим на передавання повідомлень, якщо для кожної команди записування він передає байти мережею в окремому повідомленні. За одну операцію читання приймач одержить тільки одне повідомлення. Таким чином, зберігаються межі повідомлень, що часто необхідно для обміну структурованими даними. Наприклад, у мережевій грі кожен учасник відправляє іншим гравцям пакет даних з інформацією про свою позицію.

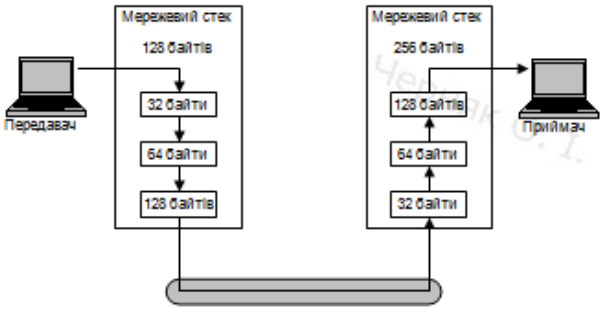


Рисунок 5.1 – Передавання дейтаграм

Протокол, який не зберігає меж повідомлень, називають протоколом, орієнтованим на передавання потоків.

Передавач безупинно передає дані: приймач зчитує стільки даних, скільки є у наявності, незалежно від меж повідомлень. Передавач може

розбивати повідомлення на частини чи поєднувати кілька повідомлень, щоб сформувати більший пакет даних. Приймач зчитує дані у мережевий стек у міру їхнього надходження і буферизує для конкретної програми. При зчитуванні програмою система надає максимально можливу кількість даних.

Об'єднання пакетів даних залежить від декількох факторів, наприклад, від максимального блоку переданої інформації чи застосування алгоритму Nagle. У відношенні TCP/IP застосування Nagle полягає у тому, що вузол накопичує дані перед відправленням і очікує, доки накопичиться досить даних чи закінчиться зазначений тайм-аут. Це потрібно, щоб партнеру не довелося пересилати пакет даних пустим – тільки з одним повідомленням. Відправлення великої кількості невеликих пакетів даних викликає істотні витрати через численні перевірки на наявність помилок і обмін підтвердженнями. З боку одержувача мережевий стек накопичує дані, які надходять, для конкретного процесу. Якщо одержувач зчитує дані, маючи 256-байтовий буфер, то всі 224 байти повертаються відразу. Якщо приймач вимагає зчитати тільки 20 байтів, система поверне тільки 20 байтів.

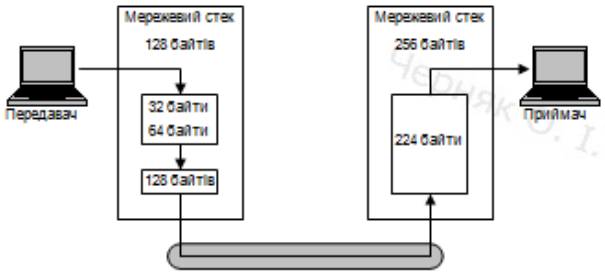


Рисунок 5.2 – Потокове передавання

Псевдопотік. При псевдопотоковому режимі передавач відправляє дані повідомленнями, а приймач отримує дані потоком. Переміщення даних псевдопотоком можна розглядати як звичайний, орієнтований на потік, протокол.

Будь-який протокол, як правило, передбачає орієнтовані і не орієнтовані на з'єднання служби.

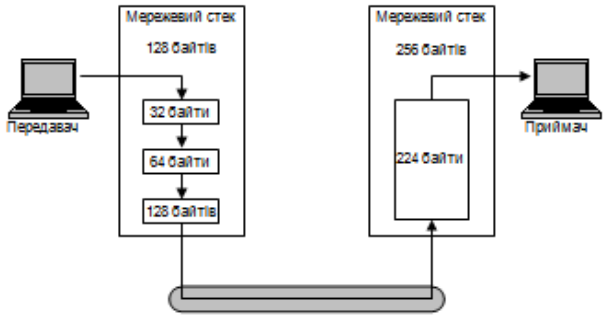


Рисунок 5.3 – Передавання у режимі псевдопотоку

5.1.2 Режими з'єднання

Протоколи, орієнтовані на з'єднання, перед будь-яким обміном даними встановлюють канал зв'язку між двома сторонами. Це гарантує існування маршруту між двома сторонами і коректний обмін інформацією. Встановлення каналу зв'язку між двома учасниками призводить до додаткових витрат. Крім того, гарантія доставки даних ще збільшує витрати через додаткові обчислення для перевірки правильності передавання.

Протоколи, не орієнтовані на з'єднання, не гарантують, що приймач дійсно приймає дані. Служби, не орієнтовані на з'єднання, схожі з поштовим зв'язком, відправник адресує лист певній людині й опускає його у поштову скриньку. Однак він не знає, чи існує взагалі одержувач листа, і чи зможе поштова служба доставити послання.

5.1.3 Надійність і порядок доставки повідомлень

У більшості випадків надійність і порядок доставки нерозривно пов'язані з тим, орієнтований протокол на з'єднання чи ні.

Надійність чи гарантована доставка означає, що кожен байт даних буде доставлений від відправника зазначеному отримувачу без змін. Ненадійний протокол не гарантує ні доставку кожного байта, ні цілісність даних.

Протокол, що зберігає порядок даних, гарантує, що приймач одержить ці дані у тому порядку, у якому вони були відправлені. Відповідно, протокол, що не зберігає порядок байтів, не дає такої гарантії.

При встановленні з'єднання сторони встановлюють канал зв'язку між собою, що гарантує надійність і порядок доставки пакетів даних. Якщо порушено порядок приймання пакетів або якісь пакети не надійшли, відбувається повторне передавання. Проте забезпечення надійності і порядку не гарантує автоматично цілісності даних.

Основна перевага протоколів, не орієнтованих на з'єднання, – це швидкість: вони не витрачають час на встановлення віртуального з'єднання з приймачем. Для невеликих повідомлень протоколи, не орієнтовані на з'єднання, швидші на порядок, ніж протоколи, орієнтовані на з'єднання. Тому дейтаграми зручні для передавання не дуже важливих даних невеликих об'ємів, наприклад, для ігрових програм: кожен гравець може відправляти дейтаграми без встановлення з'єднання, щоб періодично повідомляти про свою позицію у грі. Якщо один клієнт пропускає пакет, то він швидко одержує інший, що створює видимість безупинного зв'язку.

5.1.4 Коректне завершення роботи

Коректне завершення роботи характерне тільки для протоколів, орієнтованих на з'єднання. При цьому одна сторона ініціює завершення сеансу зв'язку, а інша – усе ще має можливість зчитувати дані, що затрималися у каналі зв'язку чи мережевому стеку. Орієнтований на з'єднання протокол, який не підтримує коректне завершення роботи, терміново завершує сеанс зв'язку, ігноруючи будь-які дані, що не були зчитані приймачем.

При використанні TCP виконується такий алгоритм коректного завершення роботи. Ініціатор завершення сеансу відправляє партнеру дейтаграму з керуючим прапорцем FIN. Одержавши цю дейтаграму партнер відправляє керуючий прапорець ACK ініціатору, щоб підтвердити одержання прапорця FIN, але усе ще може відправляти дані. Прапорець FIN означає, що ініціатор завершення сеансу відправляти дані більше не буде. Як тільки партнер завершить відправлення своїх даних, він також відправить прапорець FIN, одержання якого ініціатор підтверджує прапорцем ACK. Після цього сеанс зв'язку вважається завершеним.

5.1.5 Широкомовне передавання

Широкомовне передавання – це передавання даних з однієї робочої станції всім іншим робочим станціям ЛМ. Цю функцію мають не орієнтовані на з'єднання протоколи.

Недоліком широкомовних повідомлень є те, що їх змушений

обробляти кожен комп'ютер. Наприклад, користувач передає повідомлення всім станціям ЛМ. Мережевий адаптер кожного комп'ютера одержує повідомлення і записує його у свій мережевий стек. Стек визначає програми, які повинні одержати це повідомлення. Звичайно більшості комп'ютерів у мережі ці дані не потрібні і вони їх відкидають. Проте кожен змушений витрачати час на обробку пакета даних, щоб перевірити, чи потрібні вони для якої-небудь програми. Наслідком є високе робоче навантаження при широкомовленні, що може суттєво сповільнити роботу у ЛМ. Як правило, маршрутизатори не транслюють широкомовних пакетів даних.

5.1.6 Багатоадресне передавання

Багатоадресне передавання – це здатність одного процесу передавати дані одному чи більше одержувачам. Методика приєднання процесу до багатоадресного сеансу залежить від застосовуваного для передавання даних протоколу.

Багатоадресне передавання протоколом IP є видозміненою формою широкомовлення. Для нього необхідно, щоб усі зацікавлені у прийманні і передаванні вузли були членами особливої групи. При приєднанні процесу до групи багатоадресного передавання на мережевому адаптері додається фільтр. Він змушує мережеве устаткування обробляти і транслювати мережевим стеком до відповідного процесу тільки дані, призначені груповій адресі, до якої приєднався процес.

Багатоадресне передавання використовується, наприклад, у програмах для відеоконференцій.

5.1.7 Якість обслуговування

Керуючи якістю обслуговування (Quality of Service, QoS), програма може зарезервувати певну частину пропускної здатності мережі для монопольного використання. Наприклад, у відеоконференціях для плавності і чіткості відеодані повинні надходити з мережі рівномірно і з певною швидкістю. У недавньому минулому плавне відтворення досягалося за рахунок нагромадження відеоданих у буфері. Якщо дані передавалися нерівномірно, паузи згладжувалися кадрами з буфера, то QoS дозволяє резервувати певну частину ємності каналу зв'язку, щоб гарантувати рівномірне передавання і приймання відеоданих. Це означає, що за рахунок використання QoS теоретично програма може не буферизувати інформацію.

5.1.8 Фрагментарні повідомлення

Фрагментарні повідомлення (*partial message*) передають тільки орієнтовані на повідомлення протоколи.

Припустимо, програмі необхідно одержати повідомлення, а локальний комп'ютер прийняв лише його частину. Це звичайне явище, особливо, якщо комп'ютер відправника передає великі повідомлення. У комп'ютера приймача може не вистачити ресурсів, щоб вмістити повідомлення повністю. Насправді більшість орієнтованих на повідомлення протоколів накладають розумні обмеження на максимально можливий розмір дейтаграми, щоб така ситуація не виникала часто. Крім того, більшість дейтаграмних протоколів підтримує передавання великих повідомлень, що проходять фізичним середовищем декількома блоками. Тому, коли програма спробує прочитати повідомлення, фактично буде прийнятий лише його фрагмент.

Якщо протокол підтримує фрагментарні повідомлення, то він вказує, що повернуті дані – лише частина повідомлення. Інакше мережевий стек намагається зберегти фрагменти повідомлення доти, доки повідомлення не надійде повністю. Якщо з якої-небудь причини залишки повідомлення не будуть прийняті, більшість протоколів, що не підтримують фрагментарні повідомлення, просто відкинуть неповну дейтаграму.

5.1.9 Маршрутизація

Якщо протокол є маршрутизованим, то між двома робочими станціями можна встановити канал зв'язку (віртуально орієнтований на з'єднання або канал передавання дейтаграм), незалежно від того, яка мережева апаратура їх розділяє.

Наприклад, комп'ютер А знаходиться в окремій від комп'ютера В підмережі. Між ними розташований маршрутизатор, що з'єднує ці підмережі. Маршрутизований протокол "знає", що ці комп'ютери розташовані у різних підмережах, тому спрямовує пакет даних маршрутизатору, який вирішує, як краще переслати дані комп'ютеру В. Оскільки немаршрутизований протокол не здатний передавати дані між мережами, маршрутизатор видаляє будь-які його пакети. Маршрутизатор не пересилає пакет даних немаршрутизованого протоколу, навіть якщо його адресат знаходиться у підключеній підмережі. Єдиний немаршрутизований протокол, підтримуваний платформами Win32 –

NetBEUI.

5.1.10 Інші характеристики

Кожен протокол, підтримуваний на платформах Win32, крім основних має специфічні чи унікальні характеристики, наприклад, порядок передавання байтів чи максимально припустимий розмір пакета. Однак, далеко не всі ці характеристики важливі для розробки Winsock-програми. У Winsock 2 передбачено механізм перерахування кожного доступного постачальника протоколу й опитування його характеристик.

5.2 Мережеві протоколи, що підтримуються Win32

У табл. 5.1 перераховані основні доступні протоколи і деякі підтримувані ними режими роботи. NetBIOS підтримує відправлення дейтаграм як унікальним, так і груповим клієнтам, загальне широкомовлення не підтримується.

5.3 Мережеві протоколи у Windows CE

На відміну від інших платформ Win32, Windows CE підтримує тільки TCP/IP. Крім того, Windows CE підтримує тільки Winsock 1.1. Windows CE підтримує NetBIOS поверх TCP IP за допомогою перенаправлювача, але не дозволяє звертатися до цього протоколу ні через інтерфейс NetBIOS, ні через Winsock.

5.4 Ініціалізація Winsock

Перед викликом будь-якої функції Winsock необхідно завантажити правильну версію бібліотеки Winsock за допомогою функції ***WSAStartup***:

int WSAStartup(
WORD WersionRequested,
LPWSADATA lpWSAData);

Параметри функції.

WersionRequested – версія бібліотеки Winsock, яку необхідно завантажити. На сучасних платформах Win32 використовується версія 2.2. Єдиний виняток – Windows CE, що підтримує тільки Winsock 1.1. Для завантаження версії Winsock 2.2 потрібно вказати значення 0x0202 або макрос MAKEWORD(2,2). Верхній байт визначає додатковий номер версії,

нижній – основний.

Таблиця 5.1 – Характеристики протоколів, доступних у Windows

Протокол	Назва	Формат інформації	Встановлення з'єднання	Надійність	Порядок пакетів	Коректне завершення сеансу	Підтримка широкомовлення	Підтримка багатоадресності	QoS	Макс. розмір повідомлення (байтів)
IP	MSAFD TCP	Потік	+	+	+	+	-	-	-	Без обм.
	MSAFD UDP	Повід.	-	-	-	-	+	+	-	64KB
	RSVP TCP	Потік	+	+	+	+	-	-	+	Без обм.
	RSVP UDP	Повід.	-	-	-	-	+	+	+	64KB
IPX\ SPX	MSAFD nwln-kipx [IPX]	Повід.	-	-	-	-	+	+	-	576
	MSAFD nwln-kspx [SPX]	Повід.	+	+	+	-	-	-	-	Без обм.
	MSAFD nwln-kspx [SPX] [псевдопотік]	Повід.	+	+	+	-	-	-	-	Без обм.
	MSAFD nwln-kspx [SPXII]	Повід.	+	+	+	+	-	-	-	Без обм.
	MSAFD nwln-kspx [SPXII] [псевдопотік]	Повід.	+	+	+	+	-	-	-	Без обм.
Net-BIOS	Sequential Packets (послідовність пакетів)	Повід.	+	+	+	-	-	-	-	64KB
	Datagrams (дейтаграми)	Повід.	-	-	-	-	+	-	-	64KB
ATM	MSAFD ATM AAL5	Потік	+	-	+	-	-	+	+	Без обм.
	Native ATM (AAL5)	Повід.	+	-	+	-	-	+	+	Без обм.
Infrared Sockets	MSAFD Irda[IrDA]	Потік	+	+	+	+	-	-	-	Без обм.

lpWSAData – структура WSADATA, яка повертається після завершення виклику. Вона містить інформацію про версію Winsock, завантажену функцією WSAStartup.

Структура ***WSADATA*** описується так:

typedef struct WSAData

```
{
    WORD        wVersion;
    WORD        wHighVersion;
    char   szDescription[WSADESCRIPTION_LEN +1];
    char   szSystemStatus[WSASYS_STATUS_LEN+1];
    unsigned short      iMaxSockets;
    unsigned short      iMaxUdpDg;
    char *      lpVendorInfo;
} WSADATA, * LPWSADATA;
```

Єдина корисна інформація, що повертається у структурі WSADATA – поля wVersion і wHighVersion. Поля, що відносяться до максимальної кількості сокетів і максимального розміру UDP, потрібно отримувати з запису каталогу для конкретного протоколу.

Опис полів структури WSADATA:

wVersion – версія Winsock, що припускає використання виклику;

wHighVersion – вища версія Winsock, підтримувана завантаженою бібліотекою (як правило, те ж значення, що і wVersion)

szDescription – текстовий опис завантаженої бібліотеки;

szSystemStatus – текстовий рядок з відповідною інформацією про стан чи конфігурації;

iMaxSockets – максимальна кількість сокетів (пропустити це поле для Winsock 2 і більш пізніх версій);

iMaxUdpDg – максимальний розмір дейтаграми UDP;

lpVendorInfo – інформація про виробника (пропустити це поле для Winsock 2 і більш пізніх версій).

Після завершення роботи з бібліотекою Winsock потрібно викликати функцію ***WSACleanup*** для вивантаження бібліотеки і звільнення ресурсів. Інтерфейс функції:

```
int WSACleanup (void);
```

Для кожного виклику WSAStartup необхідно відповідно викликати WSACleanup.

5.5 Інформація про протокол

Winsock 2 дозволяє довідатися, які протоколи і з якими характеристиками встановлені на робочій станції за допомогою функції

WSAEnumProtocols:

```
int WSAEnumProtocols
(LPINT                  lpIProtocols,
LPWSAPROTOCOL_INFO      lpProtocolBuffer,
LPDWORD                 lpdwBufferLength);
```

Вона замінила функцію EnumProtocols з Winsock 1.1, яка ще застосовується у Windows CE. Єдина відмінність: WSAEnumProtocols повертає масив структур WSAPROTOCOL_INFO, а EnumProtocols – масив структур PROTOCOL_INFO, що містить менше полів, ніж структура WSAPROTOCOL_INFO (хоча інформація та ж). Структура WSAPROTOCOL_INFO визначена так:

```
typedef struct _WSAPROTOCOL_INFOW
{
    DWORD       dwServiceFlags1;
    DWORD       dwServiceFlags2;
    DWORD       dwServiceFags3;
    DWORD       dwServiceFlags4;
    DWORD       dwProviderFlags;
    GUID        ProviderId;
    DWORD       dwCatalogEntryId;
    WSAPROTOCOLCHAIN        ProtocolChain;
    int         iVersion;
    int         iAddressFamily;
    int         iMaxSockAddr;
    int         iMinSockAddr;
    int         iSocketType;
    int         iProtocol;
    int         iProtocolMaxOffset;
    int         iNetworkByteOrder;
    int         iSecurityScheme;
    DWORD       dwMessageSize;
    DWORD       dwProviderReserved;
    WCHAR       szProtocol[WSAPROTOCOL_LEN + 1];
} WSAPROTOCOL_INFOW, * LPWSAPROTOCOL_INFOW;
```

Особливо часто у структурі WSAPROTOCOL_INFO використовується поле dwServiceFlags1 – бітова маска різних атрибутів

протоколу. У наведеному далі списку перераховані бітові прапорці даного поля і дії, що ініціюються заданими прапорцями.

XP1_CONNECTIONLESS – протокол передає дані без встановлення з'єднання. Якщо прапорець не заданий – із встановленням з'єднання.

XPI_GUARANTEED_DELIVERY – протокол гарантує доставку даних одержувачу.

XP1_GUARANTEED_ORDER – протокол гарантує доставку даних у порядку їхнього відправлення без дублювання, хоча саму доставку не гарантує.

XP1_MESSAGE_ORIENTED – протокол обробляє межі повідомлень.

XP1_PSEUDO_STREAM – протокол передає повідомлення, але межі повідомлень ігноруються приймачем.

XP1_GRACEFUL_CLOSE – протокол підтримує двофазне завершення сеансу (кожна сторона повідомляється про намір іншої завершити сеанс зв'язку). Якщо цей прапорець не заданий, сеанс розривається без попередження.

XP1_EXPEDITED_DATA – протокол підтримує обмін терміновими (out-of-band) даними.

XP1_CONNECT_DATA – протокол підтримує передавання даних із запитом з'єднання.

XP1_DISCONNECT_DATA – протокол підтримує передавання даних із запитом роз'єднання.

XP1_SUPPORT_BROADCAST – протокол підтримує механізм широкомовлення.

XP1_SUPPORT_MULTIPOINT – протокол підтримує механізм багатоадресного передавання даних.

XP1_MULTIPOINT_CONTROL_PLANE – площина керування (control plane) маршрутизуєься, якщо прапорець не заданий – цього не відбувається.

XP1_MULTIPOINT_DATA_PLANE – площина даних маршрутизується, якщо прапорець не заданий – цього не відбувається.

XP1_QOS_SUPPORTED – протокол підтримує запити QoS.

XP1_UNI_SEND – протокол односпрямований і забезпечує лише відправлення даних.

XP1_UNI_RECV – протокол односпрямований і забезпечує лише

приймання даних.

XP1_IFS_HANDLES – дескриптори сокета, повернуті відновлювачем, є описувачами файлової системи IFS і можуть бути використані у таких API-функціях, як ReadFile і WriteFile.

XPI_PARTIAL_MESSAGE – прапорець MSG_PARTIAL, підтримується функціями WSASend і WSASendTo.

Для перевірки наявності певної властивості потрібно виконати логічне АБО відповідного прапорця з полем dwServiceFlags. Якщо результат операції ненульовий, протокол має дану властивість, інакше – ні.

Поле iProtocol визначає, до якого протоколу відноситься даний запис.

Поле iSocketType важливе, якщо протокол здатний працювати у різних режимах, наприклад, із установленням потокового чи дейтаграмного з'єднання.

Поле iAddressFamily дозволяє з'ясувати коректну структуру адресації, застосовувану даним протоколом. Ці три поля дуже важливі при створенні сокета для конкретного протоколу.

Найпростіше у WSAEnumProtocols задати lpProtocolBuffer=NULL і lpdwBufferLength=0. Викликавши функцію з правильним розміром буфера, можна отримати кілька структур WSAPROTOCOL_INFO.

5.6 Сокети Windows

Сокет – це описувач постачальника транспорту. У Win32 сокет відрізняється від описувача файлу, а тому поданий окремим типом – SOCKET. Сокет створюється однією з двох функцій:

```
SOCKET socket (int af, int type, int protocol);
SOCKET WSASocket (
int af,
int type,
int protocol,
LPWSAPROTOCOL_INFO  lpProtocolInfo,
GROUP g,
DWORD dwFlags);
```

Параметри функції такі:

af – визначає сімейство адрес протоколу. Наприклад, при створенні

UDP- чи TCP-сокета, af=AF_INET.

type – тип сокета для даного протоколу. Він може набувати одного із значень: SOCK_STREAM, SOCK_DGRAM, SOCK_SEQ_PACKET, SOCK_RAW, SOCK_RDM.

protocol – вказує конкретний транспорт.

У табл. 5.2 перераховані значення, використовувані у полях сімейства адрес, типу сокета і протоколу для даного мережевого транспорту.

Таблиця 5.2 – Основні параметри сокетів

Протокол		Сімейство адрес (af)	Тип сокета (type)	Тип протоколу (protocol)
Мережевий	Транспортний			
IP	TCP	AF_INET	SOCK_STREAM	IPPROTO_TCP
	UDP		SOCK_DGRAM	IPPROTO_UDP
	Прості сокети		SOCK_RAW	IPPROTO_RAW IPPROTO_IGMP
IPX/ SPX	MSAFD nwlnkipx [IPX]	AF_NS	SOCK_DGRAM	NSPROTO_IPX
	MSAFD nwlnkspx [SPX]	AF_IPX	SOCK_SEQ_PACKET	NSPROTO_SPX
	MSAFD nwlnkspx [SPX] [псевдопотік]		SOCK_STREAM	
	MSAFD nwlnkspx [SPXII]		SOCK_SEQ_PACKET	NSPROTO_SPXII
	MSAFD nwlnkspx [IPXII] [псевдопотік]		SOCK_STREAM	
NetBIOS	Послідовні пакети	AF_NETBIOS	SOCK_SEQ_PACKET	Номер LANA
	Дейтаграми		SOCK_DGRAM	
ATM	MSAFD ATM AAL5	AF_ATM	SOCK_RAW	ATMPROTO_AAL5
	Native ATM (AAL5)			
Infrared Sockets	MSAFD Irda [IrDA]	AF_IRDA	SOCK_STREAM	IRDA_PROTO_ SOCK_STREAM

Першим і найважливішим параметром є сімейство адрес. Цей параметр вказує протокол, що використовується у даний час, і обмежує значення другого і третього параметрів. Наприклад, сімейство адрес ATM (AF_ATM) дозволяє використовувати тільки прості сокети (SOCK_RAW). У свою чергу, вибір сімейства адрес і типу сокета обмежує вибір

транспортного протоколу. У такому разі, можна передати у параметрі protocol значення нуль, а система обирає постачальника транспорту, виходячи з параметрів af і type. Але це можливо лише у випадку, якщо після виклику функції WSAEnumProtocols поле dwProviderFlags структури WSAPROTOCOL_INFO, що повертається даною функцією, має значення PFL_MATCHES_PROTOCOL_ZERO.

Порти із сімейством адрес AF_NETBIOS недоступні з Winsock. При виклику WSAEnumProtocols жоден з постачальників транспорту NetBIOS не буде перерахований, навіть якщо встановлений на комп'ютері. Утім звернутися до NetBIOS можна через інтерфейс NetBIOS.

Створення IP-сокета дозволить програмам здійснювати підключення через TCP, UDP і протоколи IP. Для відкриття IP-сокета за допомогою протоколу TCP використовуються функції socket чи WSASocket із сімейством адрес AF_NET, типом сокета SOCK_STREAM, та значенням нуль поля протоколу:

```
s=socket(AF_INET, SOCK_STREAM, 0);
s=WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0,
WSA_FLAG_OVERLAPPED);
```

Щоб відкрити IP-сокет за допомогою протоколу UDP, замість SOCK_STREAM вказують тип сокета SOCK_DGRAM. Також можна відкривати сокет для зв'язку безпосередньо через IP. Для цього задають тип сокета SOCK_RAW.

5.6.1 Стани TCP

Для роботи з Winsock не обов'язково знати про стани TCP, але з їх допомогою можна краще зрозуміти, що відбувається з протоколом при викликах API-функцій Winsock. На рис. 5.4 зображено стани сокетів та переходи між ними у процесі відкриття.

Початковий стан будь-якого сокета – CLOSED. Як тільки клієнт ініціює з'єднання, серверу відправляється пакет SYN і клієнтський сокет переходить у стан SYN_SENT. Одержавши пакет SYN, сервер відправляє пакет SYN-ACK, а клієнт відповідає на нього пакетом ACK. З цього моменту клієнтський сокет переходить у стан ESTABLISHED. Якщо сервер не відправляє пакет SYN-ACK, клієнт після закінчення часу очікування повертається у стан CLOSED.

Якщо сокет сервера зв'язаний і прослухує локальну адресу та порт, то він знаходиться у стані LISTEN. При спробі клієнта встановити

з'єднання сервер одержує пакет SYN і відповідає пакетом SYN-ACK. Стан сокета сервера змінюється на SYN_RCVD. Нарешті, після відправлення клієнтом пакета ACK сокет сервера переводиться у стан ESTABLISHED.

Існує два способи закрити з'єднання. Якщо процес закриття починає програма, то закриття називається активним, інакше – пасивним. Обидва варіанти закриття сокетів зображені на рис. 5.5.

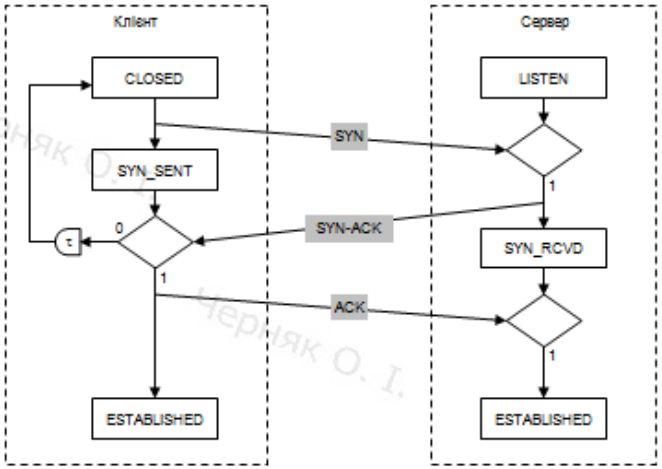


Рисунок 5.4 – Стани сокета при встановленні з'єднання

При активному закритті з'єднання програма відправляє пакет FIN. Якщо програма викликає closesocket чи shutdown (із другим аргументом SD_SEND), вона відправляє вузлу пакет FIN, і стан сокета змінюється на FIN_WAIT_1. Звичайно вузол відповідає пакетом ACK і сокет переходить у стан FIN_WAIT_2. Якщо вузол теж закриває з'єднання, він відправляє пакет FIN, а комп'ютер відповідає пакетом ACK і переводить сокет у стан TIME_WAIT.

Стан TIME_WAIT також називається станом очікування 2MSL. MSL – максимальний час життя сегмента (*Maximum Segment Lifetime*), іншими словами, час існування пакета у мережі перед його відкиданням. У кожного IP-пакета є поле часу життя (*time-to-live, TTL*). Якщо воно дорівнює нулю, то пакет можна відкинути. Кожен маршрутизатор, що

обслуговує пакет, зменшує значення TTL на 1 і передає пакет далі.

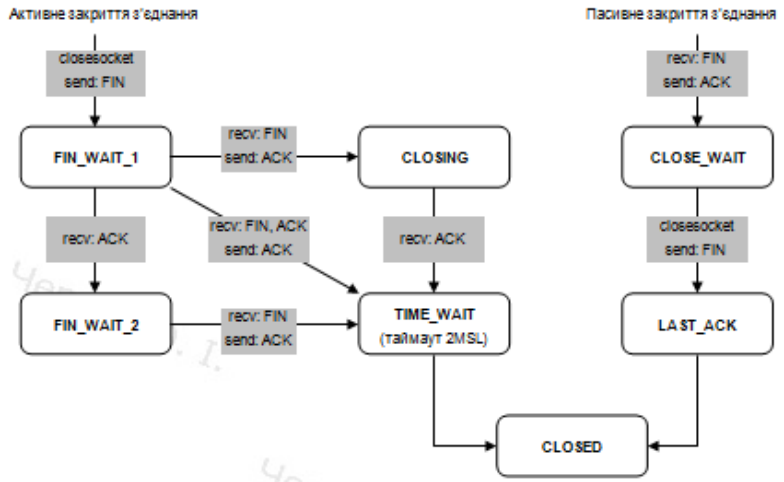


Рисунок 5.5 – Стани сокета при закритті з'єднання

Перейшовши у стан TIME_WAIT, програма залишається у ньому протягом двох періодів часу MSL. Це дозволяє TCP у випадку втрати заключного пакета ACK послати його знову, після чого відправляється сигнал FIN. Після закінчення 2MSL сокет переходить у стан CLOSED. Результат двох інших способів активного закриття – стан T1ME_WAIT. У попередньому випадку тільки одна сторона відправляла FIN і одержувала відповідь ACK, а вузол залишався вільним для передавання даних до моменту свого закриття.

Також можливі два інших способи. У першому випадку, при одночасному закритті, комп'ютер і вузол одночасно запитують закриття: комп'ютер відправляє вузлу пакет FIN і одержує від нього пакет FIN. Потім у відповідь на пакет FIN комп'ютер відправляє пакет ACK і змінює стан сокета на CLOSING. Після одержання комп'ютером пакета ACK від вузла сокет переходить у стан TIME_WAIT.

Другий випадок активного закриття є варіацією одночасного закриття: сокет зі стану FIN_WAIT_1 відразу переходить у стан TIME_WAIT. Це відбувається, якщо програма відправляє пакет FIN і

відразу після цього одержує від вузла пакет FIN-ACK. У такому випадку вузол підтверджує пакет FIN програми відправленням свого, на яке програма відповідає пакетом ACK.

Основний зміст стану TIME_WAIT полягає у тому, що доки з'єднання очікує закінчення 2MSL, сокетна пара, що бере участь у з'єднанні, не може бути використана повторно. Сокетна пара – це комбінація локального і віддаленого IP-портів. Деякі реалізації TCP не дозволяють повторно використовувати кожен з портів сокетної пари, що знаходиться у стані TIME_WAIT. У реалізації Microsoft цього дефекту немає. Утім, при спробі з'єднання із сокетною парою, що знаходиться у стані TIME_WAIT, відбудеться помилка WSAEADDRINUSE. Одне з рішень проблеми (крім чекання закінчення стану TIME_WAIT пари сокетів, що використовує локальний порт) – використовувати параметр сокета SO_REUSEADDR.

І нарешті, розглянемо пасивне закриття. Згідно з цим сценарієм програма одержує від віддаленого вузла пакет FIN і відповідає пакетом ACK. У цьому випадку сокет програми переходить у стан CLOSE_WAIT. Оскільки вузол закрив свою сторону, він більше не може відправляти дані, але програма вправі це робити, доки не закриє свою сторону з'єднання. Для закриття своєї сторони програма відправляє пакет FIN, після чого сокет програми переводиться у стан LAST_ACK. Після одержання від вузла пакета ACK сокет повертається у стан CLOSED.

5.7 Winsock і модель OSI

Постачальники транспорту з каталогу Winsock, перераховані WSAEnumProtocols, працюють на транспортному рівні моделі OSI, тобто кожний з них забезпечує обмін даними. Усі вони відносяться до якогось мережевого протоколу, що обумовлює спосіб адресації кожного вузла у мережі. Наприклад, UDP і TCP – це транспорти, хоча обоє відносяться до протоколу IP. Інтерфейс Winsock розташований між сеансовим і транспортним рівнями. Winsock дозволяє відкривати і закривати сеанс зв'язку і керувати ним для будь-якого даного транспорту. Під керуванням Windows три верхні рівні: прикладний, представницький і сеансовий, – в основному відносяться до програми Winsock. Іншими словами, програма Winsock керує всіма аспектами сеансу зв'язку і при необхідності форматує

дані.

5.8 Вибір відповідного протоколу

При розробці мережевої програми можна обрати базовий протокол з числа наявних. Однак розробляючи програму «з нуля», краще вибрати TCP/IP, оскільки цей протокол розповсюджений і широко застосовується у продуктах Microsoft. Microsoft постачає AppleTalk, NetBIOS і IPX/SPX для сумісності з іншими операційними системами й існуючими програмами.

5.9 Сімейства адрес і дозвіл імен

5.9.1 Протокол IP

Internet Protocol (IP) широко використовується в Інтернеті, підтримується більшістю ОС і застосовується як у локальних (*local area networks, LAN*), так і у глобальних мережах (*wide area networks, WAN*). IP не вимагає встановлення з'єднання і не гарантує доставку даних. Тому для передавання даних поверх IP використовуються два протоколи більш високого рівня TCP і UDP.

5.9.2 Протокол TCP

Transmission Control Protocol (TCP) реалізує зв'язок із встановленням з'єднання, забезпечує надійне безпомилкове передавання даних між двома комп'ютерами. Коли програми зв'язуються через TCP, здійснюється віртуальне з'єднання локального комп'ютера з віддаленим, після чого між ними можливий двонаправлений обмін даними.

5.9.3 Протокол UDP

Зв'язок без встановлення з'єднання виконується за допомогою User Datagram Protocol (UDP). Не гарантуючи надійності, UDP може здійснювати передавання даних безлічі адресатів і приймання даних від безлічі джерел. Дані, що відправляються клієнтом, передаються негайно, незалежно від того, чи готовий сервер до приймання. При одержанні даних від клієнта сервер не підтверджує їх приймання. Дані передаються у вигляді дейтаграм.

TCP, і UDP передають дані через IP, тому звичайно говорять про використання TCP/IP чи UDP/IP. У Winsock для IP-з'єднань передбачене сімейство адрес AF_INET, визначене у файлах Winsock.h і Winsock2.h.

5.9.4 Адресація

При використанні IP комп'ютерам призначається IP-адреса, що складається з 32 бітів, офіційно названа IP-адресою версії 4 (IPv4). Для взаємодії із сервером через TCP чи UDP клієнт повинен вказати IP-адресу сервера і номер порту служби. Щоб прослухувати вхідні запити клієнта, сервер також повинен вказати IP-адресу і номер порту. У Winsock IP-адреса і порт служби задають у структурі ***sockaddr_in***.

```
struct sockaddr_in
{
    short       sin_family;
    u_short     sin_port;
    in_addr     sin_addr;
    char        sin_zero[8];
};
```

Поля структури:

sin_family – повинно дорівнювати AF_INET. Це означає, що використовується сімейство адрес IP;

sin_port – задає комунікаційний порт;

sin_addr – зберігає IP-адреси у 4-байтовому вигляді з типом unsigned long int;

sin_zero – відіграє роль простого заповнювача, щоб структура sockaddr_in за розміром дорівнювала структурі sockaddr.

IP-адреси звичайно задають у крапковій нотації: a.b.c.d. Корисна допоміжна функція ***inet_addr*** перетворить IP-адресу з крапкової нотації у 32-бітове довге ціле без знака:

```
unsigned long inet_addr(const char *cp );
```

Параметр *cp* є рядком, що закінчується нульовим символом, тут задається IP-адреса у крапковій нотації. Ця функція як результат повертає IP-адресу, подану 32-бітовим числом з мережевим порядком проходження байтів (*network-byte order*).

5.9.5 Номери портів

При використанні TCP і UDP програма вказує, через який порт зв'язатися. Існують стандартні номери портів, зарезервовані для служб сервера, що підтримують протоколи більш високого рівня, ніж TCP. Наприклад, порт 21 зарезервований для FTP, 80 – для HTTP. Для визначення номерів портів стандартних служб, використовуються функції

getservbyname чи WSAAsyncGetServByName. Ці функції просто читають статичну інформацію з файлу з ім'ям services. Файл служб розташований у папці %WINDOWS%\System32\Drivers\Etc.

Номери портів поділяють на три категорії:

- 0-1023 – стандартні, контролюються IANA і зарезервовані для стандартних служб;
- 1024-49151 – зареєстровані IANA і можуть використовуватися процесами і програмами;
- 49152-65535 – динамічні (приватні).

Якщо при використанні функції bind програма спробує вибрати порт, уже зайнятий іншою програмою на вузлі, то система поверне помилку WSAEADDRINUSE.

5.9.6 Спеціальні адреси

Спеціальна адреса INADDR_ANY дозволяє серверній програмі слухати клієнта через будь-який мережевий інтерфейс на своєму комп'ютері. Звичайно програми сервера використовують цю адресу, щоб прив'язати сокет до локального інтерфейсу для прослуховування з'єднань. Якщо на комп'ютері декілька мережевих адаптерів, то ця адреса дозволить програмі одержувати відгуки від декількох інтерфейсів.

Спеціальна адреса – INADDR_BROADCAST дозволяє широкомовно розсилати UDP-дейтаграми IP-мережею. Для її використання необхідно у програмі задати параметр сокета SO_BROADCAST.

5.9.7 Порядок байтів

Різні процесори залежно від своєї архітектури зберігають і оброблюють числа в одному з двох порядків байтів: big-endian чи little-endian. Наприклад, процесори Intel x86 зберігають і оброблюють багатобайтові числа у порядку від менш значущого до більш значущого байта (*little-endian*). Порядок, у якому зберігаються числа у пам'яті комп'ютера, називається системним (*host-byte-order*). IP-адреса та номер порту передаються мережею у порядку від старшого байта до молодшого (*big-endian*), що називається мережевим порядком (*network-byte-order*). Є цілий ряд функцій для перетворення чисел із системного порядку у мережевий і навпаки. Чотири таких API-функції перетворять числа із системного порядку у мережевий:

```
u_long htonl(u_long hostlong);
```

*int WSAHtonl (SOCKET s, u_longl hostlong, u_long * lpnetlong);*
u_short htons(u_short hostshort);
*int WSAHtons(SOCKET s, u_short hostshort, u_short * lpnetshort);*

Параметри hostlong функцій htonl і WSAHtonl – чотирибайтові числа із системним порядком. Функція htonl повертає число з мережевим порядком, а WSAHtonl – число з мережевим порядком через параметр lpnetlong. Параметр hostshort функцій htons і WSAHtons є двобайтовим числом із системним порядком. Функція htons повертає число як двобайтове значення з мережевим порядком. Функція WSAHtons повертає число через параметр lpnetshort.

Перетворення порядку байтів з мережевого у системний виконують такі чотири функції:

u_long ntohl(u_long netlong);
*int WSANtohl (SOCKET s, u_long netlong, u_long * lphostlong);*
u_short ntohs(u_short netshort);
*int WSANtohs(SOCKET s, u_short netshort, u_short * lphostshort);*

Продемонструємо, як створити структуру sockaddr_in за допомогою вже описаних функцій inet_addr і htons:

```
sockaddr_in InternetAddr;
int nPort = 5150;
InternetAddr.sin_family = AF_INET;
InternetAddr.sin_addr.s_addr = inet_addr("136.149.3.29");
InternetAddr.sin_port = htons(nPort);
```

Тепер підготуємо сокет для з'єднання через TCP чи UDP.

5.9.8 Перетворення імен

Для підключення до вузла через IP Winsock-програма повинна знати IP-адресу цього вузла. Користувачі більш охоче звертаються до комп'ютерів за допомогою імен вузлів, що запам'ятовуються легко. У Winsock передбачено дві функції для перетворення імені в IP-адресу.

Функції gethostbyname і WSAAsyncGetHostByName відшукують у базі даних вузла відомості про вузол, що відповідають його імені. Обидві функції повертають структуру ***hostent***:

```
struct hostent
{
    char *h_name;
    char **      h_aliases;
```



```
    short        h_addrtype;
    short        h_length;
    char **      h_addr_list;
};
```

Поля структури:

h_name – є офіційним ім'ям вузла. Якщо у мережі використовується доменна система імен (*Domain Name System, DNS*), то як ім'я сервера буде повернуте повне ім'я домену (*Fully Qualified Domain Name, FQDN*). Якщо у мережі застосовується локальний файл вузлів (hosts, lmhosts), то буде повернуто перший запис після IP-адреси;

h_aliases – масив, що завершується нулем (*null-terminated array*) додаткових імен вузла;

h_addrtype – сімейство адрес, що повертається;

h_length – визначає довжину у байтах кожної адреси з поля h_addr_list;

h_addr_list – масив, що завершується нулем і містить IP-адреси вузла (вузол може мати кілька IP-адрес). Кожна адреса у цьому масиві подана у мережевому порядку. Звичайна програма використовує першу адресу з масиву. Утім, при одержанні декількох адрес, програма повинна вибирати адресу випадковим чином з числа доступних, а не використовувати першу.

API-функція ***gethostbyname*** визначена так:

*hostent * gethostbyname (const char * name);*

Параметр name – дружнє ім'я шуканого вузла. При успішному виконанні функції повертається вказівник на структуру hostent, що зберігається у системній пам'яті. У процесі роботи програми вміст структури може змінюватись системою. Оскільки структура hostent обслуговується системою, то програма не повинна звільняти повернуту структуру.

WSAAsyncGetHostByName – асинхронна версія функції gethostbyname, що оповіщає програму про завершення свого виконання за допомогою повідомлень Windows:

```
HANDLE WSAAsyncGetHostByName(
HWND hWnd,
unsigned int wMsg,
const char * name,
```

*char * buf,*

int buflen);

Параметри функції:

hWnd – дескриптор вікна, що одержить повідомлення після завершення виконання асинхронного запиту;

wMsg – Windows-повідомлення, що буде повернуто після завершення виконання асинхронного запиту;

name – дружнє ім'я шуканого вузла;

buf – вказівник на область даних, куди поміщається hostent. Цей буфер повинен бути більшим за структуру hostent і мати розмір, визначений у MAXGETHOSTSTRUCT.

Існують також дві функції пошуку інформації про вузол: gethostbyaddr і WSAAsyncGetHostByAddr. Вони дозволяють за IP-адресою вузла знайти його зрозуміле користувачу ім'я.

Функція ***gethostbyaddr*** визначена так:

*hostent * gethostbyaddr(const char * addr, int len, int type);*

Параметр addr – вказівник на IP-адресу у мережевому порядку.

Параметр len задає довжину параметра addr у байтах.

Параметр type повинен мати значення AF_NET (IP-адреса).

WSAAsyncGetHostByAddr – асинхронна версія функції gethostbyaddr.

Функція ***getservbyname***. Інтерфейс функції:

*servant * getservbyname(const char * name, const char * proto);*

Параметри функції:

name – ім'я шуканої служби. Наприклад, при визначенні порту, що використовується службою FTP, параметру name потрібно передати вказівник на рядок "ftp";

proto – посилається на рядок, що вказує протокол, під яким зареєстрована служба з параметра name.

Функція ***WSAAsyncGetServByName*** – асинхронна версія getservbyname.

У Windows 2000 застосовано новий динамічний метод реєстрації і запиту інформації про служби для TCP і UDP. Серверні програми можуть зареєструвати ім'я служби, IP-адреси і номер порту служби за допомогою функції WSASetService. Клієнтські програми запитують інформацію про ці служби за допомогою комбінації API-функцій: WSALookupSetviceBegin,

WSALookupServiceNext і WSALookupServiceEnd.

5.10 API-функції сервера

Сервер – це процес, що очікує підключення клієнтів для обслуговування їхніх запитів. Сервер повинен прослухувати з'єднання на стандартному імені. У кожного протоколу своя схема адресації, а тому і свої особливості іменування. У TCP/IP таким ім'ям є IP-адреса локального інтерфейсу і номер порту.

Перший крок встановлення з'єднання – прив'язка сокета даного протоколу до його стандартного імені функцією bind. Другий – переведення сокета у режим прослуховування функцією listen. І нарешті, сервер повинен прийняти з'єднання клієнта функцією accept чи WSAAccept.

Основні виклики функцій, що повинні зробити клієнт і сервер для встановлення каналу зв'язку між комп'ютерами у мережі, зображені на рис. 5.6.

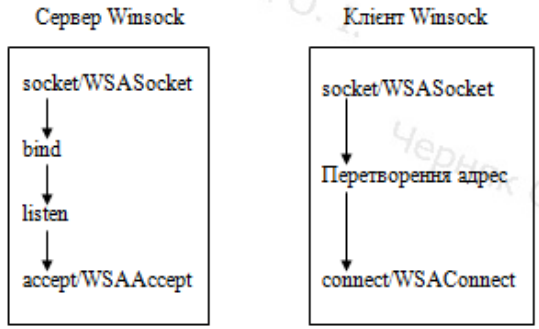


Рисунок 5.6 – Встановлення з'єднання

Розглянемо кожен API-виклик, необхідний для прив'язки, прослуховування і встановлення з'єднання з клієнтом.

5.10.1 Функції зв'язування і прослуховування сокета

Функція ***bind***. Зв'язує сокет із заданою адресою. Інтерфейс функції:

*int bind (SOCKET s, const sockaddr *name, int namelen);*

Параметри функції:

s – задає сокет, на якому будуть очікуватись запити клієнтів на з'єднання;

name – вказівник на структуру, у якій знаходиться локальна адреса. При виклику bind потрібно привести його до типу sockaddr*;

namelen – розмір переданої структури адреси.

Далі наведено приклад, що ілюструє прив'язку сокета при TCP-з'єднанні:

```
SOCKET s;
sockaddr_in addr;
int port = 5150;
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
addr.sin_family = AF_INET;
addr.sin_port = htons(port);
addr.sin_addr.S_addr = htonl(INADDR_ANY);
bind(s, (sockaddr *)&addr, sizeof(addr));
```

У даному випадку сокет вказує на IP-інтерфейс з номером порту 5150. Формально виклик bind зв'язує сокет з IP-інтерфейсом і портом.

При виникненні помилки функція bind повертає значення SOCKET_ERROR. Найпоширеніша помилка при виклику bind – WSAEADDRINUSE. У випадку використання TCP/IP це означає, що з локальним IP-інтерфейсом і номером порту вже зв'язаний інший процес, чи вони знаходяться у стані TIME_WAIT. При повторному виклику bind для вже зв'язаного сокета повертається помилка WSAEFAULT.

Функція ***listen***.

Переводить сокет у режим прослуховування. Інтерфейс функції:

```
int listen(SOCKET s, int backlog);
```

Параметри функції:

s – зв'язаний сокет;

backlog – максимальна довжина черги з'єднань, що очікують оброблення. Нехай значення цього параметра дорівнює 2, тоді при одночасному прийманні трьох клієнтських запитів перші два з'єднання будуть поміщені у чергу чекання, і програма зможе їх обробити. Третій запит поверне помилку WSAECONNREFUSED. Після того як сервер прийме з'єднання, запит видаляється з черги, а інший займає його місце. Значення backlog залежить від постачальника протоколу. Неприпустиме backlog значення замінюється найближчим дозволеним. Стандартного

способу одержати дійсне значення backlog немає.

Помилки, пов'язані з listen, досить прості. Найчастіша з них – WSAEINVAL. Звичайно це означає, що перед listen не була викликана функція bind. Іноді при виклику listen виникає помилка WSAEADDRINUSE, але частіше вона виникає при виклику bind.

5.10.2 Функції приєднання сокета

Приєднання нового сокета до прослуховуючого сокета виконується за допомогою функцій accept і WSAAccept.

Функція ***accept***:

```
SOCKET accept( SOCKET s, sockaddr *addr, int *addrlen);
```

Параметри функції:

s– зв'язаний сокет у стані прослуховування;

addr – вказівник на адресу існуючої структури sockaddr_in;

addrlen – вказівник на довжину структури sockaddr_in.

Виклик accept обслуговує перший запит, що знаходиться у черзі на з'єднання. Після його завершення структура addr буде містити інформацію про IP-адресу клієнта, що відправив запит, а параметр addrlen – розмір структури. Крім того, accept повертає новий дескриптор сокета, що відповідає прийнятому клієнтському з'єднанню. Для всіх таких операцій з цим клієнтом повинен застосовуватися новий сокет. Прослуховуючий сокет використовується для приймання інших клієнтських з'єднань і продовжує знаходитися у режимі прослуховування.

Функція ***WSAAccept*** здатна встановлювати з'єднання залежно від результату обчислення умови:

```
SOCKET WSAAccept(
SOCKET s,
sockaddr * addr,
LPINT addrlen,
LPCONDITIONPROC lpfnCondition,
DWORD dwCallbackData );
```

Перші три параметри – ті ж, що і в accept для Winsock 1.

lpfnCondition – вказівник на функцію, що викликається при запиті клієнта. Вона визначає можливість приймання з'єднання і має такий прототип:

```
int CALLBACK ConditionFunc(
LPWSABUF lpCallerId.
```

LPWSABUF lpCallerData,
LPQOS lpSQOS.
LPQOS lpGQOS,
LPWSABUF lpCalleeId,
LPWSABUF lpCalleeData,
*GROUP * g.*
DWORD dwCallbackData);

lpCallerId – містить адресу об'єкта, що з'єднується.

Структура WSABUF використовується багатьма функціями Wipsock 2 і визначена так:

```
typedef struct __WSABUF
{
    u_long len;
    char * buf;
} WSABUF,* LPWSABUF;
```

Залежно від її використання поле len визначає розмір буфера, на який посилається поле buf чи кількість даних у буфері buf.

lpCallerId – вказує на структуру WSABUF, поле buf якої вказує на структуру sockaddr, що містить адресу клієнта. Щоб одержати коректний доступ до інформації, потрібно привести вказівник buf до відповідного типу sockaddr. Більшість мережевих протоколів віддаленого доступу підтримують ідентифікацію абонента на етапі запиту.

lpCallerData – вказує на структуру, що містить дані, відправлені клієнтом у ході запиту з'єднання. Якщо ці дані не зазначені, він дорівнює NULL. Більшість мережевих протоколів, таких як TCP/IP, не використовують дані про з'єднання. Щоб довідатися, чи підтримує протокол цю можливість, потрібно викликати функцію WSAEnumProtocols.

lpSQOS і lpGQOS – задають рівень якості обслуговування, запитуваний клієнтом. Обидва параметри посилаються на структуру, що містить інформацію про вимоги пропускної здатності для приймання і передавання, Якщо клієнт не запитує параметри якості обслуговування (*quality of service, QoS*), то вони дорівнюють NULL. Різниця між ними у тому, що lpSQOS використовується для одного сокета, а lpGQOS – для груп сокетів. Групи сокетів не реалізовані і не підтримуються Winsock 1 і Winsock 2.

lpCalleeId – вказівник на структуру WSABUF, поле buf якої вказує на структуру sockaddr, що містить локальну адресу, до якої підключений клієнт. Ця інформація корисна, якщо сервер запущений на багатоадресній машині. Якщо очікуючий сервер зв'язаний з адресою INADDR_ANY, запити з'єднання будуть обслуговуватися у будь-якому мережевому інтерфейсі, а параметр lpCalleeId буде містити адресу інтерфейсу, що прийняв з'єднання.

lpCalleeData – посилається на структуру WSABUF, яку сервер може використовувати для відправлення даних клієнту у ході встановлення з'єднання. Якщо постачальник послуг підтримує цю можливість, поле len вказує максимальне число відправлення байтів. У такому випадку сервер копіює деяку, не більшу від цього значення, кількість байтів у блок buf структури WSABUF і обновляє поле len, щоб показати, скільки байтів передається. Якщо сервер не повинен повертати дані про з'єднання, то перед поверненням умовна функція приймання з'єднання присвоїть полю len значення нуль. Якщо постачальник не підтримує передавання даних про з'єднання, поле len буде дорівнює нулю. Знову ж, більшість протоколів, що підтримуються платформами Win32, не підтримують обмін даними при встановленні з'єднання.

Обробивши передані у функцію ConditionFunc параметри, сервер повинен вирішити – приймати, відхиляти чи затримати запит з'єднання. Якщо з'єднання приймається, функція ConditionFunc поверне значення CF_ACCEPT, якщо відхиляється – CF_REJECT. Якщо за якихось причин на даний момент рішення не може бути прийнято, повертається CF_DEFER.

Як тільки сервер готовий обробити запит, він викликає функцію WSAAccept. Функція ConditionFunc виконується в одному процесі з WSAAccept, і повинна працювати якнайшвидше. У протоколах, підтримуваних платформами Win32, клієнтський запит затримується, доки не буде обчислене значення функції ConditionFunc. У більшості випадків базовий мережевий стек до моменту виклику умовної функції вже може прийняти з'єднання. А при поверненні значення CF_REJECT стек просто закриває його.

При виникненні помилки повертається значення INVALID_SOCKET, найчастіше – WSAEWOULDBLOCK. Воно повертається, якщо сокет знаходиться у неблокуючому асинхронному

режимі і якщо немає з'єднання для приймання. Якщо умовна функція поверне CF_DEFER, WSAAccept генерує помилку WSATRY_AGAIN, якщо CF_REJECT – то помилку WSAECONNREFUSED.

5.11 API-функції клієнта

Клієнтська частина значно простіша і для встановлення з'єднання потрібно всього три етапи: створити сокет функцією socket чи WSASocket, перетворити ім'я сервера (залежить від використовуваного протоколу) та ініціювати з'єднання функцією connect чи WSAGonnect.

5.11.1 Функції встановлення з'єднання

Встановлення з'єднання здійснюється викликом connect чи WSAConnect.

Функція ***connect***:

int connect(
SOCKET s,
*const sockaddr * name,*
int namelen);

Параметри:

s – дійсний TCP-сокет для встановлення з'єднання;
name – структура, що містить адресу і порт віддаленого сокета;
namelen – довжина змінної name.

Функція ***WSAConnect***:

int WSAConnect(
SOCKET s,
*const sockaddr * name,*
int namelen,
LPWSABUF lpCallerData,
LPWSABUF lpCalleeData,
LPQOS lpSQOS,
LPQOS lpGQOS);

Перші три параметри такі ж, як і у функції connect.

lpCallerData – вказівник на буфер з даними, що відправляються клієнтом серверу разом із запитом на з'єднання;

lpCalleeData – вказівник на буфер з даними, що повертаються сервером під час встановлення з'єднання.

Обидві змінні є структурами WSABUF, і для lpCallerData поле len повинно вказувати довжину даних переданого буфера buf. У випадку lpCalleeData поле len визначає розмір буфера buf, куди приймаються дані від сервера.

lpSQOS і lpGQOS – вказівники на структури QoS, що визначають вимоги до пропускної здатності відправлення і приймання даних встановлюваного з'єднання. Параметр lpSQOS вказує вимогу до сокета s, а lpGQOS – до групи сокетів. На даний момент групи сокетів не підтримуються. Нульове значення lpSQOS означає, що програма не висуває вимог до якості обслуговування.

Якщо на віддаленому комп'ютері не виконується програма, що прослухує даний порт, функція connect поверне помилку WSAECONREFUSED. Інша помилка – WSAETIMEDOUT – виникає, коли адресат недоступний, наприклад, через відмову комунікаційного обладнання на шляху до вузла чи відсутності вузла у мережі.

5.12 Передавання даних

Для пересилання даних через сокет використовуються функції send та WSASend. Аналогічно, для приймання даних існують функцій recv і WSARecv.

Функції, що використовуються при відправленні і приймані даних, не призначені для роботи з кодуванням UNICODE. Відправити рядок символів UNICODE можна використовуючи тип UNICODE або привівши до типу char. У першому випадку при вказанні кількості символів, які відправляються чи прийнятих символів значення параметра, який вказує кількість переданих або прийнятих символів, повинно бути у два рази більшим, тому що кожен UNICODE-символ займає 2 байти масиву. У другому випадку потрібно спочатку перевести рядок з UNICODE у ASCII функцією WideCharToMultiByte.

Усі функції приймання і відправлення даних при виникненні помилки повертають код SOCKET_ERROR. Для одержання більш докладної інформації про помилку потрібно викликати функцію WSAGetLastError. Найпоширеніші помилки – WSAECONNABORTED і WSAECONNRESET. Обидві виникають при закритті з'єднання: або після закінчення часу очікування або при закритті з'єднання партнерським

вузлом. Ще одна типова помилка – WSAEWOULDBLOCK, як правило, виникає при використанні неблокуючих чи асинхронних сокетів. Вона означає, що функція не може бути виконана у даний момент. Функції send і WSASend призначені для відправлення даних через сокет.

Функція ***send*** визначена так

*int send(SOCKET S, const char * buf, int len, int flags);*

Параметри:

s – описувач сокета для відправлення даних;

buf – вказівник на символьний буфер, що містить дані для відправлення;

len – містить кількість символів у буфері;

flags – встановлює режим передавання. Може набувати значень 0, MSG_DONTROUTE, MSG_OOB чи результату логічного АБО над двома останніми параметрами. При вказанні прапорця MSG_DONTROUTE транспорти не будуть маршрутизувати пакети, що відправляються. Обробка цього запиту залишається на розсуд базового протоколу (наприклад, якщо транспорт не підтримує цей параметр, запит ігнорується). Прапорець MSG_OOB вказує, що дані повинні бути відправлені поза смугою (out of band), тобто терміново.

При успішному виконанні функція send повертає кількість переданих байтів, інакше – помилку SOCKET_ERROR. Одна з типових помилок – WSAECONNABORTED, відбувається при розриві віртуального з'єднання через помилку протоколу чи закінчення часу очікування. У цьому випадку сокет повинен бути закритий, тому що він більше не може використовуватися. Помилка WSAECONNRESET відбувається, якщо програма на віддаленому вузлі, виконавши апаратне закриття, розриває віртуальне з'єднання чи зненацька завершується, чи відбувається перезавантаження віддаленого вузла. У такій ситуації сокет також повинен бути закритий. Ще одна помилка – WSAETIMEDOUT, найчастіше відбувається при обриві з'єднання через збої мережі чи відмови віддаленої системи без попередження.

Функція ***WSASend*** визначена так:

int WSASend (
SOCKET s,
LPWSABUF lpBuffers,
DWORD dwBufferCount,

LPDWORD lpNumberOfBytesSent,
DWORD dwFlags,
LPWSAOVERLAPPED lpOverlapped,
LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine);

Параметри:

s – описувач сокета для відправлення даних;

lpBuffers – вказівник на структуру WSABUF чи на масив цих структур;

dwBufferCount – кількість переданих структур WSABUF. Слід пам'ятати, що структура WSABUF включає сам символьний буфер і його довжину. Це називається комплексним введенням-виведенням (*scatter-gather I/O*). При використанні декількох буферів для відправлення даних через сокет з'єднання масив буферів відправляється, починаючи з першої і закінчуючи останньою структурою WSABUF;

lpNumberOfBytesSent – вказівник на тип DWORD, що після виклику WSASend містить загальне число переданих байтів;

dwFlags – такий же, що й у функції send;

lpOverlapped і lpCompletionRoutine – використовуються для перекритого введення-виведення (*overlapped I/O*) – однієї з моделей асинхронного введення-виведення, підтримуваних Winsock.

WSASend присвоює параметру lpNumberOfBytesSent кількість записаних байтів. При успішному виконанні, функція повертає нуль, інакше – SOCKET_ERROR. Помилки ті ж, що й у функції send.

Функція ***WSASendDisconnect***

Ця спеціалізована функція використовується рідко. Вона визначена так:

int WSASendDisconnect (
SOCKET s,
LPWSABUF lpOUTboundDisconnectData);

Функція WSASendDisconnect починає процес закриття сокета і відправляє відповідні дані. Зрозуміло, вона доступна тільки для протоколів, що підтримують поступове закриття і передавання даних при його здійсненні. Жоден з існуючих постачальників транспорту на даний момент не підтримує передавання даних про закриття з'єднання. Функція WSASendDisconnect діє аналогічно shutdown з параметром SD_SEND, але

також відправляє дані, що містяться у параметрі boundDisconnectData. Після її виклику відправляти дані через сокет неможливо. У випадку невдалого завершення WSASendDisconnect повертає значення SOCKET_ERROR. Помилки, що зустрічаються при роботі функції, аналогічні помилкам send.

5.12.1 Термінові дані

Якщо програмі потрібно відправити через потоковий сокет інформацію більш високого пріоритету, вона може позначити цю інформацію як термінові дані *(out-of-band, OOB*). Програма з іншої сторони з'єднання одержує й обробляє OOB-дані через окремий логічний канал, концептуально незалежний від потоку даних.

У TCP передавання OOB-даних реалізовано шляхом додавання однобітового маркера (URG) і 16-бітового вказівника у заголовку сегмента TCP, що дозволяють виділити важливі байти в основному трафіку. На даний момент для TCP існують два способи виділення термінових даних. У RFC 793, що описує TCP і концепцію термінових даних, говориться, що вказівник терміновості у заголовку TCP є додатним зсувом байта, що розташований за байтом термінових даних. Однак у RFC 1122 цей зсув трактується як вказівник на сам байт терміновості.

У специфікації Winsock під терміном OOB розуміють як незалежні від протоколу OOB дані, так і реалізацію механізму передавання термінових даних у TCP. Для перевірки, чи є у черзі термінові дані, потрібно викликати функцію ioctlsocket з параметром SIOCATMARK.

У Winsock передбачено кілька способів передавання термінових даних. Можна вставити їх у звичайний потік або, відключивши цю можливість, викликати окрему функцію, що повертає тільки термінові дані. Параметр SO_OOBINLINE керує поведінкою OOB-даних.

У ряді випадків термінові дані використовують програми Telnet і Rlogin. Утім краще уникати застосування термінових даних – вони не стандартизовані і можуть мати інші реалізації на відмінних від Win32 платформах. Якщо потрібно час від часу передавати терміново якусь інформацію, то треба створити окремий керуючий сокет для термінових даних, а основне з'єднання надати для звичайного передавання даних.

5.12.2 Функції recv і WSARecv

Функція *recv* – основний інструмент приймання даних через сокет. Вона визначена так:

int recv(
SOCKET s,
*char * buf,*
int len,
int flags);

Параметри:

s – визначає сокет для приймання даних;

buf – символьний буфер, призначений для отриманих даних;

len – кількість прийнятих байтів чи розмір буфера buf;

flags – може приймати значення 0, MSG_PEEK, MSG_OOB чи результат логічного АБО над двома останніми параметрами. Нуль означає відсутність дій. Прапорець MSG_PEEK вказує, що доступні дані повинні копіюватися у приймальний буфер і при цьому залишатися у системному буфері. Після завершення функція повертає кількість байтів у системному буфері. Зчитувати повідомлення з використанням прапорця MSG_PEEK не рекомендується. Мало того, що через два системні виклики (один – для зчитування даних, і інший, без прапорця MSG_PEEK, – для знищення даних), знижується продуктивність. У ряді випадків цей спосіб просто ненадійний. Обсяг даних, що повертається, може не відповідати їх сумарній доступній кількості. До того ж, зберігаючи дані у системних буферах, система залишає все менше пам'яті для розміщення наступних вхідних даних. Як наслідок, зменшується розмір вікна TCP для всіх процесів-відправників, що не дозволяє програмі досягти максимальної продуктивності. Найкраще скопіювати всі дані у власний буфер і обробляти їх там. Прапорець MSG_OOB вже обговорювався раніше при розгляді відправлення даних.

Використання recv у сокетах, орієнтованих на передавання повідомлень чи дейтаграм, має кілька особливостей. Якщо при виклику recv розмір даних, що очікує оброблення, більший від наданого буфера, то після його повного заповнення виникає помилка WSAEMSGSIZE. Помилка перевищення розміру повідомлення відбувається тільки при використанні протоколів, орієнтованих на передавання повідомлень. Потокові протоколи буферизують дані, що надходять, і при запиті програмою надають їх у повному обсязі, навіть якщо кількість даних, що очікує оброблення, більше розміру буфера. Таким чином, помилка WSAEMSGS1ZE не може виникнути при роботі з потоковими

протоколами.

Функція ***WSARecv*** має додаткові, порівняно з recv, можливості: підтримує перекрите введення-виведення і фрагментарні дейтаграмні повідомлення.

int WSARecv(
SOCKET s,
LPWSABUF lpBuffers,
DWORD dwBufferCount,
LPDWORD lpNumberOfBytesRecvd,
LPDWORD lpFlags,
LPWSAOVERLAPPED lpOverlapped,
LPWSAOVERLAPPED_COMPLETION_ROUTINE #
lpCompletionRoutine);

Параметри:

s – сокет з'єднання;

lpBuffers і dwBufferCount – визначають буфери для приймання даних;

lpBuffers – вказівник на масив структур WSABUF;

dwBufferCount – визначає кількість таких структур у масиві;

lpNumberOfBytesReceived – у випадку завершення операції одержання даних вказує на кількість прийнятих цим викликом байтів;

lpFlags – може набувати значення MSG_PEEK, MSG_OOB, MSG_PARTIAL чи результату логічного АБО над будь-якими з цих параметрів.

У прапорця MSG_PARTIAL залежно від способу використання можуть бути різні значення і зміст. Для протоколів, орієнтованих на передавання повідомлень, цей прапорець встановлюється після виклику WSARecv (якщо все повідомлення не може бути повернуте через недостачу місця у буфері). У цьому випадку кожен такий виклик WSARecv встановлює прапорець MSG_PARTIAL, доки повідомлення не буде прочитане повністю. Якщо цей прапорець передається як вхідний параметр, операція приймання даних повинна завершитися, як тільки дані будуть доступні, навіть якщо це тільки частина повідомлення. Прапорець MSG_PARTIAL використовується тільки з протоколами, орієнтованими на передавання повідомлень. Запис кожного протоколу у каталозі Winsock містить прапорець, що вказує на підтримку цієї можливості. Параметри

lpOverlapped і lpCompletionROUTINE застосовуються в операціях перекритого введення-виведення.

Функція ***WSARecvDisconnect*** обернена до WSASendDisconnect і визначена так:

int WSARecvDisconnect(
SOCKET s,
LPWSABUF lpInboundDisconnectData);

Як і у WSASendDisconnect, першим її параметром є описувач сокета з'єднання.

Другий параметр – недійсна структура WSABUF для приймання даних.

Функція отримує тільки дані про закриття з'єднання, відправлені з іншої сторони функцією WSASendDisconnect, її не можна використовувати для приймання звичайних даних. До того ж, відразу після прийняття даних вона припиняє приймання з віддаленої сторони, що еквівалентно виклику shutdown з параметром SD_RECV.

Функція ***WSARecvEx*** є спеціальним розширенням Microsoft для Winsock 1. Вона ідентична recv в усьому, крім того, що параметр flags передається за посиланням. Це дозволяє процесу приймача задавати прапорець MSG_PARTIAL.

*int PASCAL WSARecvEx(SOCKET s, char * buf, int len, int *flags);*

Якщо отримані дані є лише частиною повідомлення, то у параметрі flags повертається прапорець MSG_PARTIAL. Він використовується тільки з тими протоколами, що орієнтовані на передавання повідомлень. Коли при прийнятті неповного повідомлення прапорець MSG_PARTIAL передається як частина параметра flags, функція WSARecvEx завершується негайно, повертаючи прийняті дані. Якщо у буфері не вистачає місця, щоб прийняти повідомлення повністю, WSARecvEx поверне помилку WSAEMSGSIZE, а решта даних, що не змогли бути прийняті, будуть відкинуті. Зверніть увагу на різницю між прапорцем MSG_PARTIAL і помилкою WSAEMSGSIZE: якщо виникла помилка, то це означає, що повідомлення надійшло повністю, однак вхідний буфер занадто малий для того, щоб його прийняти.

У WSARecvEx також можна використовувати прапорці MSG_PEEK і MSG_OOB.

5.13 Особливості передавання потоків даних

Більшість протоколів із встановленням з'єднання між предавачем і приймачем є потоковими. Важливо враховувати, що при використанні будь-якої функції відправлення чи приймання даних через потоковий сокет немає гарантії, що буде прочитано чи записано весь замовлений обсяг даних. Нехай потрібно відправити 2048 байтів із символьного буфера функцією send:

```
char sendbuff[2048];
int nBytes = 2048;
// Заповнення буфера sendbuff 2048 байтами даних
// Присвоєння s значення дійсного потокового сокета з'єднання
ret = send(s, sendbuff, nBytes, 0);
```

Функція send повідомить про відправлення менше 2048 байтів. Змінна ret буде містити кількість переданих байтів. При відправленні даних внутрішні буфери утримують їх до моменту відправлення безпосередньо через дріт. Причиною неповного відправлення може бути, наприклад, передавання великої кількості даних, при цьому всі буфери занадто швидко заповняться.

У TCP/IP також існує так званий розмір вікна. Приймаюча сторона регулює його, вказуючи кількість даних, які здатна прийняти. При переповненні даними одержувач може задати нульовий розмір вікна, щоб справитися з даними, що надійшли. Це призведе до призупинення відправлення даних, доки розмір вікна не стане більшим нуля. У нашому випадку розмір буфера може дорівнювати 1024 байтам, отже, потрібно повторно відправити решту 1024 байти. Відправлення всього вмісту буфера забезпечить такий фрагмент програми:

```
char sendbuff[2048];
int nBytes = 2048, nLeft, idx;
// Заповнення буфера sendbuff 2048 байтами даних
// Присвоєння s значення дійсного потокового сокета з'єднання
nLeft = nBytes;
idx = 0;
while (nLeft > 0)
{
    ret = send(s, &sendbuff[idx], nLeft, 0);
```



```
    if (ret == SOCKET_ERROR)
    {
        // Помилка
    }
    nLeft -= ret;
    idx += ret;
}
```

Це справедливо і при потоковому отриманні даних, але не так важливо. Наприклад, якщо у всіх повідомлень однаковий розмір (512 байтів), то прочитати їх можна так:

```
char recvbuff[1024];
int ret, nLeft, idx;
nLeft = 512;
idx = 0;
while (nLeft > 0)
{
    ret = recv(s, &recvbuff[idx], nLeft, 0);
    if (ret == SOCKET_ERROR)
    {
        // Помилка .
    }
    idx += ret;
    nLeft-=ret;
}
```

Ситуація ускладниться, якщо розмір повідомлень буде змінюватись. Тоді потрібно реалізувати власний протокол, що повідомляє одержувачу про розмір отриманого повідомлення. Наприклад, перші 4 байти повідомлення можуть містити його розмір.

5.13.1 Комплексне введення-виведення

Вперше принцип комплексного введення-виведення (Scatter-Gather I/O) був застосований у функціях recv і write сокетів Берклі (Berkeley Sockets). У Winsock 2 його підтримують функції WSARecv, WSARecvFrom, WSASend, WSASendTo. Комплексне введення-виведення найчастіше використовують програми, що відправляють і приймають дані у специфічному форматі. Наприклад, якщо передані клієнтом серверу повідомлення повинні складатися з фіксованого 32-байтового заголовка,

що визначає деякі дії, 64-байтного блоку даних і закінчуватися 16-байтовою контрольною сумою. У цьому випадку функція WSASend може бути викликана для масиву відповідних трьох структур WSABUF. На приймаючій стороні одним із вхідних параметрів викликуваної функції WSARecv також повинні бути три структури WSABUF, що містять 32, 64 і 16 байтів.

При використанні потокових сокетів операції комплексного введення-виведення інтерпретують кілька буферів даних як один безупинний. Функція прийняття даних може завершитися раніше, ніж будуть заповнені всі буфери. У сокетах, орієнтованих на передавання повідомлень, кожна операція одержання даних приймає одне повідомлення, довжина якого не більша розміру буфера. Якщо у буфері недостатньо місця, виклик закінчується помилкою WSAEMSGSIZE і дані урізаються до розміру доступного простору. У протоколах, що підтримують фрагментарні повідомлення, для запобігання втрати даних можна використовувати прапорець MSG_PARTIAL.

5.13.2 Завершення сеансу

Після закінчення роботи із сокетом необхідно викликати функцію closesocket для закриття з'єднання і звільнення всіх ресурсів, зв'язаних з описувачем сокета, однак, її неправильне використання може привести до втрати даних. Тому перед викликом closesocket потрібно коректно завершити сеанс. Правильно написана програма повідомляє одержувача про закінчення відправлення даних. Так само повинен вчинити і віддалений комп'ютер. Така поведінка називається коректним завершенням сеансу і здійснюється за допомогою функції shutdown.

Функція ***shutdown***:

int shutdown(SOCKET s, int how);

Параметр how може набувати значення SD_RECEIVE, SD_SEND чи SD_BOTH. Значення SD_RECEIVE забороняє всі такі виклики будь-яких функцій приймання даних, на протоколи нижнього рівня це не діє. Якщо у черзі TCP-сокета є дані або вони надходять пізніше, з'єднання все одно закривається. UDP-сокети в аналогічній ситуації продовжують приймати дані і ставити їх у чергу. SD_SEND забороняє всі виклики функції відправлення даних. У випадку TCP-сокетів після підтвердження одержувачем приймання усіх відправлених даних передається пакет FIN. Нарешті, SD_BOTH забороняє як приймання, так і відправлення.

Функція ***closesocket*** закриває сокет. Вона визначена так:

int closesocket (SOCKET s);

Після виклику closesocket всі подальші операції із сокетом закінчаться помилкою WSAENOTSOCK. Якщо не існує інших посилань на сокет, то усі зв'язані з дескриптором ресурси будуть звільнені, включаючи дані у черзі.

Асинхронні виклики, які очікують надходження даних, скасовуються без повідомлення. Операції, які очікують перекритого введення-виведення, також анулюються. Усі події, які виконуються у даний момент, процедура і порт завершення, зв'язані з перекритим введенням-виведенням, завершаться помилкою з повідомленням WSA_OPERATION_ABORTED.

5.13.3 Приклади

У більшості програм для приймання інформації використовують тільки функції recv чи WSARecv, а для відправлення – функції send чи WSASend. Інші функції забезпечують додаткові можливості і рідко використовуються (чи підтримуються тільки транспортними протоколами).

Розглянемо приклад клієнт-серверної взаємодії з урахуванням описаних принципів і функцій. У лістингу 5.1 наведено код простого сервера. Програма створює сокет, прив'язує його до локального IP-інтерфейсу і порту та слухає з'єднання клієнта. Після приймання від клієнта запиту на з'єднання створюється новий сокет, через який відбувається обмін інформацією.

Лістинг 5.1 – Приклад сервера TCP, що приймає з'єднання від одного клієнта

```
// Ім'я модуля: Server1.cpp
//
// VC++6.0: Додати у меню "Project-Setting-Library" запис ws2_32.lib.
// VC++2005: Додати у меню "Project-...Propeties-ConfigurationPropeties-
Linker-Input-//AdditionalDependecies" запис ws2_32.lib.
//
#include "stdafx.h"
#include "winsock2.h"
#include "windows.h"
```

```
#include <stdio.h>
#include <conio.h>
#define PORT 5555
#define BUFFLEN 4096
char Msg[]="Hi from Server1!";
int main()
{
        WSADATA Wsadata;
        SOCKET ListenSock, ClientSock;
        sockaddr_in ClientAddr, LocalAddr;
        int AdrLen, RetCod;
        char Buffer[BUFFLEN];
        WsaStartup(0x0202, &Wsadata);
        ListenSock=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        LocalAddr.sin_family=AF_INET;
        LocalAddr.sin_addr.S_un.S_addr=htonl(INADDR_ANY);
        LocalAddr.sin_port=htons(PORT);
        bind(ListenSock, (sockaddr*)&LocalAddr, sizeof(LocalAddr));
        listen(ListenSock, 8);
        AddrLen=sizeof(ClientAddr);
        ClientSock=accept(ListenSock, (sockaddr*)&ClientAddr, &AddrLen);
        printf("Connect client. Addr: %s.\n", inet_ntoa(ClientAddr.sin_addr));
        RetCod=recv(ClientSock, Buffer, BUFFLEN,0);
        Buffer[RetCod]=0;
        puts("Message from client: ");
        puts(Buffer);
        send(ClientSock, Msg, sizeof(Msg), 0);
        getch();
        CloseSocket(ListenSock);
        CloseSocket(ClientSock);
        WSACleanup();
        return 0;
}
```

У лістингу 5.2 наведено приклад сервера, що виконує під'єднання багатьох клієнтів завдяки використанню потоків.

Лістинг 5.2 – Приклад сервера TCP, що приймає з'єднання від багатьох клієнтів

```
// Ім'я модуля: Server2.cpp
//
// VC++6.0: Додати у меню "Project-Setting-Library" запис ws2_32.lib.
// VC++2005: Додати у меню "Project-...Propeties-ConfigurationPropeties-
Linker-Input-//AdditionalDependecies" запис ws2_32.lib.
//
#include "stdafx.h"
#include "winsock2.h"
#include "windows.h"
#include <stdio.h>
#include <conio.h>
#define PORT;dcm 55555
#define BUFFLEN 4096
char Msg[]="Hi from Server2!";
int ClientNom=0;
sockaddr_in ClientAddr;
DWORD WINAPI ClientThread(LPVOID lpParam);
int main()
{
        HANDLE hThread;
        int ThreadID;
        WSADATA Wsadata;
        SOCKET ListenSock, ClientSock;
        sockaddr_in LocalAddr;
        WSAStartup(0x0202, &Wsadata);
        ListenSock=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        LocalAddr.sin_family=AF_INET;
        LocalAddr.sin_addr.S_un.S_addr=htonl(INADDR_ANY);
        LocalAddr.sin_port=htons(PORT);
        bind(ListenSock, (sockaddr*)&LocalAddr, sizeof(LocalAddr));
        listen(ListenSock, 8);
        while(!kbhit())
        {
            int AddrLen;
```

```
        AddrLen=sizeof(ClientAddr);
        ClientSock=accept(ListenSock, (sockaddr*)&ClientAddr,
        &AddrLen);
        CreateThread(NULL, 0, ClientThread, (LPVOID)ClientSock, 0,
        (LPDWORD)&ThreadID);
        CloseHandle(hThread);
    }
    return 0;
}
DWORD WINAPI ClientThread(LPVOID lpParam)
{
    int Nom=ClientNom++;
    printf("CONNECT to client %d with addr:\n%s\n", Nom,
    inet_ntoa(ClientAddr.sin_addr));
    DWORD RetCod;
    SOCKET s=(SOCKET)lpParam;
    char Buffer[BUFFLEN];
    RetCod=recv(s, Buffer, BUFFLEN,0);
    Buffer[RetCod]=0;
    printf("MESSAGE from client %d:\n", Nom);
    puts(Buffer);
    getch();
    send(s, Msg, sizeof(Msg),0);
    closesocket(s);
    return 0;
}
```

Програма клієнта TCP із встановленням з'єднання подана у лістингу 5.3. Спочатку клієнт створює сокет за допомогою виклику функції *socket*. Потім за допомогою виклику функції *connect* клієнт з'єднується із сервером. Як тільки з'єднання встановлене, клієнт за допомогою виклику функції *send* відправляє серверу повідомлення. Після відправлення повідомлення клієнт очікує відповіді сервера і отримує її за допомогою виклику функції *recv*. Всі отримані через сокет дані виводяться на екран.

Лістинг 5.3 – Приклад клієнта TCP

```
//Ім'я модуля: Client.cpp
//
// VC++6.0: Додати у меню "Project-Setting-Library" запис ws2_32.lib.
// VC++2005: Додати у меню "Project-...Propeties-ConfigurationPropeties-
Linker-Input-//AdditionalDependecies" запис ws2_32.lib.
//
#include "stdafx.h"
#include "winsock2.h"
#include "windows.h"
#include <stdio.h>
#include <conio.h>
#define PORT 55555
#define BUFFLEN 4096
char IPAddr[]="127.0.0.1";
char Msg[]="Hi from Client!";
int main()
{
    WSADATA Wsadata;
    SOCKET s;
    sockaddr_in ServerAddr;
    int RetCod;
    char Buffer[BUFFLEN]="";
    WSAStartup(0x0202, &Wsadata);
    s=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    ServerAddr.sin_family=AF_INET;
    ServerAddr.sin_port=htons(PORT);
    ServerAddr.sin_addr.S_un.S_addr=inet_addr(IPAddr);
    if(connect(s, (sockaddr*)&ServerAddr,
    sizeof(ServerAddr))==SOCKET_ERROR)
    {
        puts("ERROR! Can't connect to Server.");
        getch();
        return 1;
    }
    printf("CONNECT to server with addr:\n%s.\n",
```

```
        inet_ntoa(ServerAddr.sin_addr));
        getch();
        send(s, Msg, sizeof(Msg),0);
        RetCod=recv(s, Buffer, sizeof(Buffer),0);
        Buffer[RetCod]=0;
        puts("MESSAGE from Server: ");
        puts(Buffer);
        getch();
        closesocket(s);
        WSACleanup();
        return 0;
}
```

5.14 Протоколи, що не потребують з'єднання

Принцип дії таких протоколів інший, тому що у них використовуються інші методи відправлення і приймання даних. У таких протоколах архітектурою обміну є не клієнт-сервер, а приймач-передавач. Різниця між приймачем і передавачем полягає тільки у тому, хто викликає функції приймання, а хто – передавання інформації. Причому функція приймання завжди повинна викликатись першою. Будь-який з варіантів функції приймання виконується у блокуючому режимі і очікує отримання даних.

5.14.1 Приймач

Спочатку створюють сокет функцією socket чи WSASocket. Потім виконують прив'язку сокета до локальної адреси функцією bind (як і у випадку протоколів, орієнтованих на сеанси). Різниця у тому, що не можна викликати listen та accept. Замість цього потрібно просто передавати або отримувати дані. Оскільки у цьому випадку з'єднання немає, приймаючий сокет може одержувати дейтаграми від будь-якої машини у мережі.

Найпростіша функція приймання – ***recvfrom***:

```
int recvfrom(
SOCKET s,
char * buf,
int len,
int flags,
```



```
sockaddr * from,
int * fromlen );
```

Перші чотири параметри такі ж, як і для функції recv, включаючи допустимі значення для flags: 0, MSG_OOB і MSG_PEEK;

from – структура sockaddr для даного протоколу слухаючого сокета, на розмір структури адреси посилається fromlen.

Після повернення виклику структура sockaddr буде містити адресу робочої станції, що відправляє дані.

У Winsock 2 для приймання інформації без встановлення з'єднання застосовується ***WSARecvFrom***:

```
int WSARecvFrom(
SOCKET s,
LPWSABUF lpBuffers,
DWORD dwBufferCount,
LPDWORD lpNumberOfBytesRecvd,
LPDWORD lpFlags,
sockaddr * lpFrom,
LPINT lpFromlen,
LPWSAOVERLAPPED lpOverlapped,
LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine );
```

Функції можна надати один чи кілька буферів WSABUF, вказавши їхню кількість у dwBufferCount – у цьому випадку можливе комплексне введення-виведення. Сумарна кількість зчитаних байтів передається у lpNumberOfBytesRecvd. При виклику функції WSARecvFrom, lpFlags може приймати такі значення: 0 (при відсутності параметрів), MSG_OOB, MSG_PEEK чи MSG_PARTIAL. Дані прапорці можна комбінувати логічною операцією АБО. Якщо при виклику функції задано прапорець MSG_PARTIAL, постачальник перешле дані навіть у випадку приймання лише частини повідомлення. Після повернення прапорець задається у MSG_PARTIAL, тільки якщо повідомлення прийняте частково. Після повернення WSARecvFrom присвоїть параметру lpFrom (вказівник на структуру sockaddr) адресу комп'ютера-відправника. Знову ж, параметр lpFromLen вказує на розмір структури SOCKADDR, однак у даній функції він є вказівником на DWORD. Два останніх параметри – lpOverlapped і lpCompletionROUTINE – використовуються для перекритого введення-

виведення.

Інший спосіб приймання-відправлення даних у сокетах, що не потребують з'єднання, – встановлення з'єднання (хоч це і звучить дивно). Після створення сокета і зв'язування з локальною адресою можна викликати connect чи WSAConnect, присвоївши параметру sockaddr адресу віддаленого комп'ютера, з яким необхідно зв'язатися. Фактично ніякого з'єднання не відбувається. Адреса сокета, переданого у функцію з'єднання, асоціюється із сокетом, щоб було можна використовувати функції recv і WSARecv замість recvfrom чи WSARecvFrom (оскільки джерело даних відоме). Якщо програмі потрібно одночасно зв'язуватися лише з однією кінцевою точкою, використовується можливість підключення сокета дейтаграм.

5.14.2 Передавач

Є два способи відправлення даних через сокет, при якому не потрібне з'єднання. Перший і найпростіший – створити сокет і викликати функцію sendto чи WSASendTo.

Розглянемо функцію ***sendto***:

int sendto(
SOCKET s,
*const char * buf,*
int len,
int flags,
*const sockaddr * to,*
int tolen);

Параметри цієї функції такі ж, як і у recvfrom, за винятком buf – буфера даних для відправлення, і tolen – показує скільки байтів відправляти.

Параметр to – вказівник на структуру sockaddr з адресою приймаючої робочої станції.

Також можна використовувати функцію ***WSASendTo*** з Winsock 2:

int WSASendTo(
SOCKET s,
LPWSABUF lpBuffers,
DWORD dwBufferCount,
LPDWORD lpNumberOfBytesSent,
DWORD dwFlags,

*const sockaddr * lpTo,*
int iToLen,
LPWSAOVERLAPPED lpOverlapped,
LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine);

Функція WSASendTo аналогічна своїй попередниці. Вона приймає вказівник на одну чи кілька структур WSABUF з даними для відправлення одержувачу у вигляді параметра lpBuffers, а dwBufferCount задає кількість структур.

Для комплексного введення-виведення можна відправити кілька структур WSABUF. Перед виходом WSASendTo присвоює четвертому параметру – lpNumberOfBytesSent – кількість реально відправлених одержувачу байтів. Параметр lpTo – структура sockaddr для даного протоколу з адресою приймача. Параметр iToLen – довжина структури sockaddr. Два останніх параметри – lpOverlapped і lpCompletionRoutine – використовуються для перекритого введення-виведення.

Як і при отриманні даних, сокет, що не потребує з'єднання, можна підключати до адреси кінцевої точки і відправляти дані функціями send і WSASend. Після створення цього прив'язування не можна використовувати sendto і WSASendTo з іншою адресою – буде видано помилку WSAEISCONN.

5.14.3 Особливості протоколів, що не потребують з'єднання

Протоколи, що потребують з'єднання, орієнтовані на передавання повідомлень. Тому потрібно враховувати такі фактори.

По-перше, оскільки орієнтовані на передавання повідомлень протоколи зберігають межі повідомлень, то дані, поставлені у чергу для відправлення, блокуються до завершення виконання функції відправлення. Якщо відправлення не може бути завершене, то при асинхронному або неблокуючому режимі введення-виведення функція відправлення поверне помилку WSAEWOULDBLOCK. Це означає, що операційна система не змогла обробити дані і потрібно викликати функцію відправлення повторно. Важливим є те, що в орієнтованих на повідомлення протоколах запис відбувається тільки внаслідок самостійної дії.

По-друге, при виклику функції приймання потрібно надати місткий буфер, інакше функція видасть помилку WSAEMSGSIZE: буфер заповнений, і дані, що залишилися, відкидаються. Винятком є протоколи,

що підтримують обмін фрагментарними повідомленнями, наприклад AppleTalk PAP. Якщо була прийнята лише частина повідомлення, функція WSARecvEx присвоює параметру flag значення MSG_PARTIAL.

Для передавання дейтаграм на основі протоколів, що підтримують фрагментарні повідомлення, використовують одну з функцій WSARecv чи recv. Після чергового виклику recv може бути отримана лише частина дейтаграми. Проте, при виклику recv не можна відстежити повноту зчитування повідомлення (це задача програміста). Через дане обмеження зручніше використовувати функцію WSARecvEx, що дозволяє задавати і зчитувати прапорець MSG_PARTIAL. Функції Winsock 2 WSARecv і WSARecvFrom також підтримують роботу з даним прапорцем.

Якщо замість bind викликається sendto чи WSASendTo, чи спочатку встановлюється з'єднання, мережевий стек автоматично вибирає локальну IP-адресу з таблиці маршрутизації. Таким чином, якщо спочатку була виконана прив'язка, початкова IP-адреса може бути неправильною і не відповідати інтерфейсу, з якого фактично була відправлена дейтаграма.

5.14.4 Звільнення ресурсів сокета

Оскільки з'єднання не встановлюється, його формального розриву чи коректного закриття не потрібно. Після припинення відправлення чи одержання даних відправником, чи одержувачем просто викликається функція closesocket з описувачем необхідного сокета, у результаті чого звільняються усі виділені йому ресурси.

5.14.5 Приклади

Розглянемо відправлення і приймання дейтаграм.

Приймання дейтаграм просте. Спочатку необхідно створити сокет, потім прив'язати його до локального інтерфейсу. Для прив'язки до інтерфейсу за замовчуванням визначте його IP-адресу функцією getsockname. Як параметр їй передається сокет, а вона повертає структуру sockaddr_in, яка вказує зв'язаний із сокетом інтерфейс. Потім для читання вхідних даних залишається тільки виконати виклики recvfrom. Ми використовуємо recvfrom, тому що нас не цікавлять фрагментарні повідомлення, оскільки протокол UDP не підтримує їх передавання. Фактично, стек TCP/IP намагається зібрати велике повідомлення з отриманих фрагментів. Якщо один чи кілька фрагментів відсутні або порушено порядок їхнього проходження, стек відкидає все повідомлення. У лістингу 5.4 наведений код приймача.

Лістинг 5.4 – Приймач, який не вимагає встановлення з'єднання

```
// Ім'я модуля: Receiver.cpp
//
// VC++6.0: Додати у меню "Project-Setting-Library" запис ws2_32.lib.
// VC++2005: Додати у меню "Project-...Propeties-ConfigurationPropeties-
Linker-Input-//AdditionalDependecies" запис ws2_32.lib.
//
#include "stdafx.h"
#include "winsock2.h"
#include "stdio.h"
#include <conio.h>
#include <windows.h>
#define PORT 55555
#define ADDR "127.0.0.1"
#define BUFFLEN 256
char Buffer[BUFFLEN];
int main()
{
        WSADATA wsd;
        SOCKET s;
        int RetCod;
        sockaddr_in SenderAddr,ReceiverAddr;
        WSAStartup(0x0202,&wsd);
        s=socket(AF_INET,SOCK_DGRAM,0);
        SenderAddr.sin_addr.S_un.S_addr=inet_addr(ADDR);
        SenderAddr.sin_port=htons(PORT);
        SenderAddr.sin_family=AF_INET;
        ReceiverAddr.sin_addr.S_un.S_addr=inet_addr(ADDR);
        ReceiverAddr.sin_port=htons(PORT);
        ReceiverAddr.sin_family=AF_INET;
        bind(s,(sockaddr*)&ReceiverAddr,sizeof(ReceiverAddr));
        RetCod=recvfrom(s,Buffer,BUFFLEN,0,(sockaddr*)&SenderAddr,
        &AddrLen);
        Buffer[RetCod]=0;
        puts(Buffer);
        getch();
```

```
    return 0;
}
```

Замість функції recvfrom, яка вказує адресу відправника, можна використовувати функцію recv, якщо перед нею викликати функцію connect:

```
connect(s,(sockaddr*)&SenderAddr,sizeof(SenderAddr));
RetCod=recv(s,Buffer,BUFFLEN,0);
```

При цьому з'єднання між клієнтом і сервером не встановлюється, а лише виконується прив'язування до сокета адреси, за якою у подальшому відбувається відправлення. В обох випадках можна приймати повідомлення від будь-якого комп'ютера, якщо у структурі SenderAddr у полі S_addr задати спеціальну адресу.

```
SenderAddr.sin_addr.S_un.S_addr=htonl(INADDR_ANY);
```

У лістингу 5.5 наведено код відправника, який не потребує з'єднання.

Лістинг 5.5 – Відправник, який не потребує встановлення з'єднання

```
// Ім'я модуля: Sender.c
//
// VC++6.0: Додати у меню "Project-Setting-Library" запис ws2_32.lib.
// VC++2005: Додати у меню "Project-...Propeties-ConfigurationPropeties-
Linker-Input-AdditionalDependecies" – запис ws2_32.lib.
//
#include "stdafx.h"
#include "winsock2.h"
#include "stdio.h"
#include <conio.h>
#include <windows.h>
#define PORT 55555
#define ADDR "127.0.0.1"
char Msg[]="Hello from Sender";
int main()
{
    WSADATA wsd;
    SOCKET s;
```



```
    sockaddr_in SenderAddr,ReceiverAddr;
    WSAStartup(0x0202,&wsd);
    s=socket(AF_INET,SOCK_DGRAM,0);
    SenderAddr.sin_addr.S_un.S_addr=inet_addr(ADDR);
    SenderAddr.sin_port=htons(PORT);
    SenderAddr.sin_family=AF_INET;
    ReceiverAddr.sin_addr.S_un.S_addr=inet_addr(ADDR);
    ReceiverAddr.sin_port=htons(PORT);
    ReceiverAddr.sin_family=AF_INET;
    bind(s,(sockaddr*)&SenderAddr,sizeof(SenderAddr));
    sendto(s,Msg,sizeof(Msg),0,(sockaddr*)&ReceiverAddr,
    sizeof(ReceiverAddr));
    getch();
    return 0;
}
```

У іншому випадку замість функції sendto можна використовувати функцію send, якщо перед нею викликати функцію connect:

```
connect(s,(sockaddr*)&ReceiverAddr,sizeof(ReceiverAddr));
send(s,Msg,sizeof(Msg),0);
```

5.15 Додаткові функції Winsock 2

Функції WSAAddressToString і WSAStringToAddress у Winsock 2 забезпечують незалежний від протоколу спосіб перетворення структури sockaddr протоколу у форматований рядок символів і навпаки. Оскільки ці функції не залежать від протоколу, транспортний протокол повинен підтримувати перетворення рядків. У даний час ці функції працюють тільки для сімейств адрес AF_INET і AF_ATM.

Функція ***WSAAddressToString*** визначена так:

```
int WSAAddressToString(
LPSOCKADDR lpsaAddress,
DWORD dwAddressLength,
LPWSAPROTOCOL_INFO lpProtocolInfo,
OUT LPTSTR lpszAddressString,
IN OUT LPDWORD lpdwAddressStringLength);
```

Параметри:

lpsaAddress – вказівник на структуру sockaddr для конкретного протоколу, що містить адресу, яку потрібно перетворити на рядок;

dwAddressLength – розмір структури першого параметра для кожного протоколу;

lpProtocolInfo – постачальник протоколу. Постачальників протоколу можна знайти функцією WSAEnumProtocols. Якщо параметру задається значення NULL, виклик використовує постачальника першого протоколу, який підтримує сімейство адрес, вказане у параметрі lpsaAddress;

lpszAddressString – буфер, де зберігається зручна для читання адреса;

lpdwAddressStringLength – розмір lpszAddressString. При виведенні у ньому повертається довжина рядка, фактично скопійована у lpszAddressString. Якщо наданий буфер малий, функція видає помилку WSAEFAULT, а у параметрі lpdwAddressStringLength повернеться необхідний розмір у байтах.

Функція ***WSAStringToAddress***, перетворить зручну для читання адресу у структуру sockaddr:

int WSAStringToAddress(
LPTSTR AddressString,
int AddressFamily,
LPWSAPROTOCOl_INFO lpProtocolInfo,
LPSOCKADDR lpAddress,
LPINT lpAddressLength);

Параметри функції:

AddressString – адреса у такому форматі:

• для IP – XXX.XXX.XXX.XXX:Y, де X є октетом у рядку IP-адреси, а Y – номер порту;

• для ATM – NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN, де 40 символів N є 20-байтовою ATM-адресою у шістнадцятковому форматі;

AddressFamily – тип сімейства адрес для параметра AddressString.

lpProtocolInfo – постачальник протоколу. Якщо присвоїти йому NULL, то Winsock буде шукати перший доступний постачальник протоколу для типу адресного сімейства з AddressFamily. Для вибору конкретного постачальника отримують за допомогою WSAEnumProtocols

список доступних постачальників протоколу, встановлених у системі;

lpAddress – структура sockaddr, що приймає інформацію в адресному рядку;

lpAddressLength – розмір структури sockaddr, яка містить результат.

Функція ***getpeername***. Повертає інформацію про адресу підключеного партнера:

int getpeername(
SOCKET s,
*sockaddr * name,*
int FAR namelen);*

Параметри функції:

s – сокет для з'єднання;

name – вказівник на пусту структуру sockaddr;

namelen – довжина структури sockaddr.

Для сокетів дейтаграм дана функція записує адресу, передану виклику з'єднання (за винятком адрес, переданих у виклик sendto чи WSASendTo).

Функція ***getsockname***. Ця функція обернена до getpeemame і повертає адресну інформацію локального інтерфейсу певного сокета:

int getsockname(
SOCKET s,
*sockaddr * name,*
int FAR namelen);*

Використовуються ті ж параметри, що і для getpeemame, однак повертається інформація про локальну адресу. У випадку TCP адреса збігається із сокетом сервера, що слухає на заданому порту і IP-інтерфейсі.

Функція ***WSADuplicateSocket***. Дана функція застосовується для створення структури WSAPROTOCOL_INFO, яку можна передати іншому процесу, що дозволить йому відкрити описувач того ж базового сокета й оперувати даним ресурсом. Така необхідність виникає тільки між процесами. Потоки в одному й тому ж процесі можуть вільно передавати описувачі сокета. Функція визначена так:

int WSADuplicateSocket(
SOCKET s,
DWORD dwProcessId,
LPWSAPROTOCOL_INFO lpProtocolInfo);

Параметри функції:

s – описувач сокета для копіювання;

dwProcessId – код процесу, що буде використовувати скопійований сокет;

lpProtocolInfo – вказівник на структуру WSAPROTOCOL_INFO, що містить інформацію, необхідну цільовому процесу для відкриття копії описувача сокета. У деяких випадках процеси повинні взаємодіяти між собою таким чином, щоб поточний процес мав можливість передавати структуру WSAPROTOCOL_INFO цільовому процесу, який потім міг би використати її для створення описувача сокета (за допомогою функції WSASockef).

Описувачі в обох сокетах можна використовувати для введення-виведення незалежно, однак Winsock не забезпечує контроль за доступом. Тому програмісту необхідно передбачити деякі способи синхронізації. Вся інформація про стан будь-якого сокета зберігається в одному місці для всіх його описувачів, тому що у функцію копіюються описувачі сокета, а не сам сокет. Наприклад, значення будь-якого параметра сокета, заданого для одного з описувачів викликом функції setsockopt, можна потім отримати, викликавши функцію getsockopt для будь-якого іншого його описувача. Якщо процес викликає closesocket для закривання копії сокета, описувач копії у даному процесі звільняється, однак сокет залишиться відкритим, доки функція closesocket не буде викликана для останнього його описувача.

Існують певні особливості сокетів при використанні функцій WSAAsyncSelect і WSAEventSelect. Дані функції використовуються при асинхронному введенні-виведенні. При їхньому виклику з кожним із спільних описувачів скасовується реєстрація будь-якої попередньої події для сокета, незалежно від того, який із спільних описувачів використовувався для реєстрації. Тому, наприклад, через спільний для процесів A і B сокет не можна передавати події FD_READ процесу A і події FD_WRITE процесу B. Якщо необхідно передавати повідомлення про події через обидва описувача, потрібно використовувати потоки замість процесів.

Функція ***TransmitFile***. Це розширення Microsoft дозволяє швидко передавати дані з файлу. Висока ефективність обумовлена тим, що все передавання даних відбувається у режимі ядра. Якщо зчитується блок

даних з файлу, а потім викликається send чи WSASend, то відбуваються багаторазові переключення між режимами ядра і користувача. При виклику TransmitFile весь процес читання і відправлення виконується у режимі ядра. Функція визначена так:

```
BOOL TransmitFile(
SOCKET hSocket,
HANDLE hFile,
DWORD nNumberOfBytesToWrite,
DWORD nNumberOfBytesPerSend,
LPOVERLAPPED lpOveflapped,
LPTRANSMIT_FILE_BUFFERS lpTransmitBuffers,
DWORD dwFlags);
```

Параметри функції:

hSocket – підключений сокет, яким буде переданий файл;

hFile – описувач відкритого файлу;

nNumberOfBytesToWrite – кількість записуваних з файлу байтів. Якщо цей параметр дорівнює нулю, файл відправляється повністю;

nNumberOfBytesPerSend – розмір блоків даних, які відправляються, для операцій записування. Наприклад, якщо розмір дорівнює 2048, TransmitFile передасть файл порціями по 2 Кбайти, якщо нуль – використовується стандартний розмір відправлення;

lpOverlapped – вказівник на структуру OVERLAPPED, що використовується при перекритому введенні-виведенні;

lpTransmitBuffers – вказівник на структуру TRANSMIT_FILE_BUFFERS з даними, що потрібно відправити до і після передавання файлу:

```
typedef struct _TRANSMIT_FILE_BUFFERS
{
    PVOID Head;
    DWORD HeadLength;
    PVOID Tail;
    DWORD TailLength;
} TRANSMIT_FILE_BUFFERS;
```

Поля структури:

Head – вказівник на дані, що відправляються перед передаванням файлу;

HeadLength – кількість заздалегідь переданих даних;

Tail – посилається на дані, що відправляються після передавання файлу;

TailLength – кількість переданих потім байтів;

dwFlags – режими роботи TransmitFile. Прапорці можуть мати такі значення:

TF_DISCONNECT – ініціює закриття сокета після передавання даних;

TF_REUSE_SOCKET – дозволяє повторно використовувати у функції AcceptEx описувач сокета як клієнтського сокета;

TF_USE_DEFAULT_WORKER і TF_USE_SYSTEM_THREAD – вказують, що передавання повинно виконуватись у контексті стандартного системного процесу. Ці прапорці корисні при передаванні великих файлів;

TF_USE_KERNEL_APC – вказує, що передавання повинно виконуватись ядром за допомогою асинхронних викликів процедур (Asynchronous Procedure Call, APC). Це істотно збільшує продуктивність, якщо для зчитування файлу у кеш потрібна лише одна операція читання;

TF_WRITE_BEHIND – вказує, що TransmitFile може завершитися, не одержавши підтверджень про приймання даних від віддаленої системи.

5.16 Контрольні питання

1. Назвіть режими передавання даних, що використвуються у транспортних протоколах. Опишіть кожен з режимів.
2. Назвіть режими з'єднання, що використовуються у транспортних протоколах. Опишіть кожен з режимів.
3. Що означає надійність і порядок доставки у транспортних протоколах?
4. Для яких протоколів і як виконується коректне завершення роботи?
5. Опишіть режими широкомовного і багатоадресного передавання даних. Які протоколи їх підтримують?
6. Що таке якість обслуговування і фрагментарні повідомлення?
7. Назвіть мережеві і транспортні протоколи, що підтримуються Windows.
8. Назвіть характеристики транспортних протоколів мережі IP.

9. Назвіть характеристики транспортних протоколів мережі IPX/SPX.
10. Назвіть характеристики транспортних протоколів мережі NetBIOS.
11. Дайте порівняльну характеристику протоколів TCP і UDP.
12. Що таке сокет? Якими параметрами характеризується сокет? Які переваги має використання сокетів порівняно з іншими мережевими технологіями?
13. Яка функція ініціалізує бібліотеку сокетів?
14. Опишіть поля структури WSADATA.
15. Яка функція створює сокет? Опишіть її параметри.
16. Які характеристики мають сокети сімейства AF_NETBIOS?
17. Як відбувається адресація протоколу IP? Будова IP-адреси. Спеціальні адреси.
18. Опишіть поля структури sockaddr_in.
19. Що таке порт? Як поділяються номери портів?
20. Який порядок байтів в адресах і портах? Назвіть функції, які задають порядок байтів, та опишіть їхні параметри.
21. Які функції служать для перетворення адрес? Які параметри мають дані функції?
22. Які послідовності функцій повинні викликати клієнт і сервер?
23. Опишіть функції, які використовуються для зв'язування і прослуховування сокета. Опишіть їхні параметри.
24. Опишіть функції, що використовуються для приєднання сокета, і їхні параметри.
25. Опишіть функції, які використовуються для встановлення з'єднання, і їхні параметри.
26. Опишіть функції, які використовуються для потокового передавання даних, та їхні параметри.
27. Опишіть функції, які виконують термінове передавання даних мережею.
28. Опишіть функції, які використовуються для потокового приймання даних, і їхні параметри.
29. Як можна запобігти переповненню буферів при потоковому передаванні даних?
30. Що таке комплексне введення-виведення даних і як воно

реалізовано у сокетах?

31. Які функції потрібно викликати для завершення сеансу зі встановленням з'єднання? Опишіть їхні параметри.

32. Напишіть програму TCP-сервера, що обмінюється інформацією з одним клієнтом.

33. Напишіть програму TCP-сервера, що обмінюється інформацією з багатьма клієнтами.

34. Напишіть програму TCP-клієнта.

35. Який принцип дії програм, що використовують протоколи без встановлення з'єднання?

36. Які функції використовуються для дейтаграмного передавання інформації. Опишіть їх параметри.

37. Які функції використовуються для дейтаграмного отримання інформації. Опишіть їх параметри.

38. Для чого при передаванні і отриманні дейтаграм використовують встановлення з'єднання? Які інші функції при цьому використовуються?

39. Як забезпечується звільнення ресурсів після завершення обміну інформацією без встановлення з'єднання?

40. Напишіть програму передавача, що не встановлює з'єднання.

41. Напишіть програму приймача, що не встановлює з'єднання.

42. Які функції забезпечують незалежний від протоколу спосіб перетворення структури sockaddr у форматований рядок символів і навпаки? Опишіть їх параметри.

43. Які функції повертають адресу локального чи віддаленого комп'ютера? Опишіть іхні параметри.

44. Яка функція використовується для створення структури WSAPROTOCOL_INFO? Як у подальшому використовується дана структура?

45. Яка функція дозволяє швидко передавати дані з файлу на інший комп'ютер? За рахунок чого підвищується швидкість передавання? Опишіть параметри даної функції.

46. Які характеристики мають сокети сімейства AF_INET?

47. Які характеристики мають сокети сімейства AF_IPX?

48. Які характеристики мають сокети сімейства AF_NETBIOS?

ПІСЛЯМОВА

У даному посібнику описано широкий спектр технологій прикладного мережевого програмування в операційній системі Windows. Розглянуто перенаправлювач, поштові скриньки, іменовані канали та інтерфейс NetBIOS. Найбільшу увагу приділено інтерфейсу мережевого програмування Winsock.

Надано інформацію про перенаправлювач Windows, який дозволяє додаткам одержувати доступ до ресурсів файлової системи Windows мережею. Описано, як за допомогою перенаправлювача здійснюється обмін інформацією у мережі і як при цьому використовується система безпеки Windows NT і Windows 2000.

Розглянуто принципи мережевого програмування з використанням поштових скриньок. Розглянуто переваги і недоліки даного підходу.

Детально розглянуто використання для мережевого програмування технології іменованих каналів. Іменовані канали надають просту архітектуру клієнт-сервер для надійного передавання даних. Для передавання даних мережею даний інтерфейс використовує перенаправлювач Windows. Основна перевага іменованих каналів – вони дозволяють скористатися вбудованими можливостями захисту операційної системи Windows.

Розглянуто NetBIOS – потужний прикладний інтерфейс мережевого програмування. Одна з його сильних сторін – незалежність від протоколу: додатки можуть працювати через TCP/IP, NetBEUI, IPX/SPX. NetBIOS дозволяє здійснювати обмін як із встановленням логічного з'єднання, так і без нього. Значна перевага NetBIOS перед сокетами – єдиний спосіб вирішення і реєстрації імен. Додаток NetBIOS потребує тільки імені NetBIOS, тоді як додаток Winsock, реалізуючи різні протоколи, повинен знати схему адресації кожного з них.

Найбільшу увагу приділено популярній технології мережевого програмування – сокетам. У рамках створення WinAPI-додатків розглянуто технологію Winsock. Winsock – це інтерфейс прикладного програмування мережевих взаємодій, реалізований на всіх платформах Win32. Інтерфейс Winsock розроблено на основі Berceley (BSD) Sockets на платформах UNIX, що працює з багатьма мережевими протоколами. У середовищі Win32 даний інтерфейс став повністю незалежним від

мережевих протоколів. У даному розділі описано характеристики протоколів, що підтримуються Win32 і, у свою чергу, підтримують інтерфейс Winsock. Описано режими сокетів, алгоритм встановлення з'єднання та функції, необхідні для встановлення з'єднання і виконання обміну.

На даний час технології мережевого програмування отримали подальший розвиток. У рамках програмного середовища Visual C++ фірма Microsoft пропонує просту технологію використання сокетів на основі сокетів MFC. Бібліотека MFC є об'єктно-орієнтованою надбудовою над WinAPI, призначеною для спрощення програмування за рахунок використання класів, що інкапсулюють функції більш високого рівня. У MFC сокет – це об'єкт відповідного класу. Існує два класи сокетів MFC: CAsyncSocket і похідний від нього CSocket. Клас CAsyncSocket містить функції, що виконують обмін інформацією між комп'ютерами у асинхронному режимі, а клас CSocket призначений для обміну у синхронному режимі. Отримати детальну інформацію про технологію програмування за допомогою сокетів MFC можна у [4, 5, 6, 7].

Програмування за допомогою сокетів в UNIX-подібних системах описано у [8].

Для професійного програмування мережевого обміну інформацією потрібно більш детальне вивчення протоколу TCP/IP. Для цього можна порекомендувати [3].

Останнім часом широкої популярності набула нова платформа програмування Microsoft, що отримала назву ".NET" (DOT NET). У рамках даної платформи розроблено нову мову програмування під назвою C#. Мова C# створена спеціально для мережевого і розподіленого програмування. На даний час мова C# набула широкого розповсюдження і вважається найбільш перспективною. C# є модифікацією мови C++, тому для вивчення цієї мови найкраще мати досвід з програмування на C++ [7]. Тим, хто знайомий з C++, для вивчення мови C# можна порекомендувати [9]. Розробка програм мовою C# для платформи DOT NET описана у [10]. У даній книзі описано також аспекти мережевого програмування. Для більш детального вивчення технології програмування у мережах з використанням сокетів та інших засобів платформи DOT NET можна порекомендувати [11, 12].

БІБЛІОГРАФІЧНИЙ ОПИС

1. Джонс Э., Оланд Дж. Программирование в сетях Microsoft Windows. Мастер-класс. : Пер. с англ. – Спб. : Питер; М.: Издательско-торговый дом "Русская редакция", 2005.– 608 с. : ил.

2. Джонсон М. Харт Системное программирование в среде Windows, 3-е издание. : Пер. с англ. – М. : Издательский дом "Вильямс", 2005. – 592 с. : ил.

3. Microsoft Corporation. Microsoft TCP/IP. Учебный курс: Официальное пособие Microsoft для самостоятельной подготовки. : Пер. с англ. – 2-е изд., испр. – М. : Издательско-торговый дом "Русская редакция", 1999. – 344 с. : ил.

4. Грегори К. Использование Visual C++ 6. Специальное издание.: Пер. с англ. – К. : Диалектика, 2000. – 816 ст. : ил.

5. Николенко Д. В. Самоучитель по Visual C++ 6.0. – СПб. : "Наука и техника", 2001. – 367 с. : ил.

6. Нортон П., Макгрегор Р. Руководство Питера Нортона. Программирование в Windows 95/NT4 с помощью MFC. В 2-х книгах. Книга 2. : Пер. с англ. – М. : "СК Пресс", 1998. – 560 с. : ил., CD-ROM.

7. Круглински Д., Уингоу Ст., Шеферд Дж. Программирование на Microsovt Visual C++ 6.0 для профессионалов. : Пер. с англ. – СПб : Питер; М.: Издательско-торговый дом "Русская редакция", 2001. – 864 с. : ил.

8. Снейдер Й. Эффективное программирование TCP/IP. Библиотека программиста. : Пер. с англ. – СПб. : "Питер", 2001. – 320 с. : ил.

9. Нэш Т. C# 2008: ускоренный курс для профессионалов. : Пер. с англ. – М. : ООО "И. Д. Вильямс", 2008. – 576 с. : ил.

10. Нейгел К., Ивьен Б., Глинн Д., Уотсон К., Скиннер М. C# 2008 и платформа .NET 3.5 для профессионалов. : Пер. с англ. – М. : ООО "И. Д. Вильямс", 2009. – 1392с. : ил.

11. Кровчик Э., Кумар В., Лагари Н., Мунгале А., Паркер Т., Шивакумар Ш., Нагел К. .NET. Сетевое программирование для профессионалов. : Пер. с англ. – М. : Издательство "ЛОРИ", 2007. – 400 с. : ил.

12. Лёве Дж. Создание служб Windows Communication Foundation. – СПб. : Питер, 2008. – 592 с. : ил.

ГЛОСАРІЙ

анонімний – *anonymous*
архітектура широкомовлення – *broadcasting architecture*
блок мережевого керування – *network control block, NCB*
блок повідомлення сервера – *server message block, SMB*
віддалені пристрої – *removed devices*
відкрите з'єднання систем – *open systems interconnect, OSI*
вільний стан – *nonsignaled state*
глобальна мережа – *wide area networks, WAN*
гніздо Windows – *Windows socket*
дескриптори безпеки – *security descriptors*
доменна система імен – *domain name system, DNS*
зайнятий стан – *signaled state*
запис керування доступом – *access control entities, ACE*
запит введення-виведення – *input-output query*
захист від несанкціонованого доступу – *access protection*
зв'язок між процесами – *interprocess communication*
ідентифікатор безпеки – *security identifier, SID*
ідентифікація – *identification*
іменовані канали – *named pipes*
інтерфейс прикладного програмування – *application programming interface, API*
клієнт для мереж Microsoft – *client for microsoft networks*
комп'ютерна мережа – *computer network*
комплексне введення-виведення – *scatter-gather I/O*
консольні додатки – *console applications*
локальна адреса – *local address*
локальна мережа – *local area networks, LAN*
максимальний час життя сегмента – *maximum segment lifetime*
маркер доступу – *access tokens*
масив, що завершується нулем – *null-terminated array*
мережева операційна система – *network operating system,* NOS
мережеве програмування – *network programming*
мережевий порядок проходження байтів – *network-byte order*
мережевий постачальник – *network provider*

мережевий протокол – *network protocol*
мережевий транспортний протокол – *network transport protocol*
мережеві додатки – *network applications*
номер мережевого адаптера – *local area network adapter number, LANA*
односторонній обмін – *one-directional exchange*
операційна система – *operation system*
основний транспортний протокол дейтаграмного обміну даними у мережі – *user datagram protocol,UDP*
основний транспортний протокол потокового обміну даними у мережі – *transmission control protocol, TCP*
основні реквізити входу – *primary login*
перекрите введення-виведення – *overlapped I/O*
перенаправлення введення-виведення – *I/O redirection*
перенаправлювач – *redirector*
персоналізація – *impersonation*
повернуте повне ім'я домену – *fully qualified domain name, FQDN*
повідомлення, дейтаграма – *datagram*
порядок від молодшого до старшого байта – *little-endian*
порядок від старшого байта до молодшого – *big-endian*
поставник декількох UNC – *multiple unc provider,MUP*
потік байтів – *byte stream*
потік повідомлень – *message stream*
поштові скриньки – *mailslots*
протокол спільного використання файлів на основі блоків повідомлень сервера – *server message block file sharing protocol*
реквізити сеансу – *session login*
системний порядок – *host-byte-order*
системний список керування доступом – *system access control list, SACL*
список вибіркового керування доступом – *discretionary access control list, DACL*
термінові дані – *out-of-band, OOB*
файлова система – *file system*
файлова система іменованих каналів – *named pipe file system, NPFS*
фрагментарне повідомлення – *partial message*
час життя – *time-to-live, TTL*
якість обслуговування – *quality of service, QoS*

Навчальне видання

Азаров Олексій Дмитрович
Черняк Олександр Іванович

ПРИКЛАДНЕ ПРОГРАМУВАННЯ У КОМП'ЮТЕРНИХ МЕРЕЖАХ

Навчальний посібник

Редактор Т. Старічек

Оригінал-макет підготовлено О. Черняком

Підписано до друку
Формат 29,7×42 ¼ . Папір офсетний.
Гарнітура Times New Roman.
Друк різографічний Ум. друк. арк.
Наклад прим. Зам. №

Вінницький національний технічний університет,
науково-методичний відділ ВНТУ.
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, к. 2201.
Тел. (0432)59-87-36.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.

Віддруковано у Вінницькому національному технічному університеті
в комп'ютерному інформаційно-видавничому центрі.
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, ГНК, к. 114.
Тел. (0432)59-81-59
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.