# Лабораторна робота №1

# Створення власної утиліти Whois.

У основі мережевих технологій лежить еталонна модель OSI (Open System Interconnection, взаємодія відкритих систем), яка була розроблена Міжнародною організацією по стандартизації. Модель OSI розділяє різні процеси, що відбуваються під час сеансу зв'язку, на сім функціональних рівнів. Кожен рівень визначається сервісом, який він надає вищестоящому рівню, і протоколом - набором правил і форматів даних для взаємодії між собою об'єктів одного рівня, що працюють на різних комп'ютерах.

1. **Фізичний рівень** - передача бітів по фізичних каналах (коаксіальний кабель, вита пара, оптоволоконний кабель). Тут визначаються характеристики фізичних середовищ і параметри електричних сигналів.
2. **Канальний рівень** - передача кадру даних між будь-якими вузлами в мережах типової топології або сусідніми вузлами довільної топології. Як адреси на канальному рівні використовуються МАС-адреси.
3. **Мережевий рівень** - доставка пакету будь-якому вузлу в мережах довільної топології. На цьому рівні немає ніяких гарантій доставки пакету.
4. **Транспортний рівень** - доставка пакету будь-якому вузлу з будь-якою топологією мережі і заданим рівнем надійності доставки. На цьому рівні є засобу для встановлення з'єднання, буферизації, нумерації і впорядковування пакетів.
5. **Сеансовий рівень** - управління діалогом між вузлами. Забезпечена можливість фіксації активної на даний момент сторони.
6. **Рівень представлення даних** - тут можливо задати перетворення даних (шифрування, стиснення).
7. **Прикладний рівень** - набір мережевих сервісів (FTP, E-mail і ін.) для користувача і додатку.

Перші три рівні забезпечуються устаткуванням, таким як мережеві карти, маршрутизатори, концентратори, мости і ін. Останні три забезпечуються операційною системою або додатком. Четвертий рівень є проміжним.

Протокол по цій моделі працює таким чином. Все починається з прикладного рівня. Пакет потрапляє на цей рівень і до нього додається заголовок. Після цього прикладний рівень відправляє цей пакет на наступний рівень (рівень уявлення). Тут йому також додається свій власний заголовок, і пакет відправляється далі. Так до фізичного рівня, який займається безпосередньо передачею даних і відправляє пакет в мережу.

Інша машина, одержавши пакет, починає зворотний відлік. Пакет з фізичного рівня потрапляє на канальний. Канальний рівень прибирає свій заголовок і піднімає пакет вище (на рівень мережі). Рівень мережі прибирає свій заголовок і піднімає пакет вище. Так пакет піднімається до рівня додатку, де залишається чистий пакет без службової інформації, яка була прикріплена на початковому комп'ютері перед відправкою пакету.

Передача даних не обов'язково повинна починатися з сьомого рівня. Якщо використовуваний протокол працює на четвертому рівні, то процес передачі почнеться з нього, і пакет підніматиметься вгору до фізичного рівня для відправки. Кількість рівнів в протоколі визначає його потреби і можливості при передачі даних.

Чим нижче знаходиться протокол (ближче до прикладного рівня), тим більше у нього можливостей і більше накладних витрат при передачі даних (більше і складніше заголовок). Протоколи, що розглядаються в роботі, знаходитимуться на різних рівнях, тому матимуть різні можливості.

Корпорація Microsoft пішла своїм шляхом і реалізувала модель OSI в TCP/IP по-своєму: принцип залишений той же, хоча змінилися назви і кількість рівнів. У Microsoft замість семи рівнів є тільки чотири. Один рівень Microsoft може виконувати все, що в OSI роблять три рівні. Наприклад, рівень додатку у Microsoft виконує все, що роблять рівень додатки, рівень уявлення і рівень сеансу разом узяті. На мал. 1.1 графічно зіставлена модель MS TCP і довідкова модель OSI. Зліва вказані назви рівнів по методу MS, а справа - рівні OSI. У центрі показані протоколи.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | MS TCP/IP | | | Справочная модель OSI | |
| Рівень додатку | Сокети Windows | | NetBios  NetBios на основі TCP/IP | | Прикладний рівень  Рівень представлення даних |
|  | Інтерфейс TDI | | | | Сеансовий рівень |
| Транспортний рівень | UDP | | TCP | | Транспортний рівень |
| Міжмережевий рівень | ICMP | | ARP | | Мережевий рівень |
| IP | | | |
| IGMP | | RARP | |
|  | Інтерфейс NDIS | | | |  |
| Рівень мережевого інтерфейсу | Internet | Драйвери мережевих карт | РРР  Трансляція пакету даних | | Канальний рівень |
| FDDI | Мережеві карти | Фізичний рівень |

**Мал.1.1. Модель OSI і варіант від MS**

**Мережеві протоколи - протокол IР**

Протокол TCP/IP - це не один цілий, а два різні протоколи, які працюють спільно. Різниця в цих протоколах дуже важлива.

Якщо подивитися на схему мережевої моделі (рис.1.1), то можна побачити, що протокол IP знаходиться на мережевому рівні. З цього можна зробити висновок, що IP виконує мережеві функції - доставка пакету будь-якому вузлу в мережах довільної топології.

Протокол IP при передачі даних не встановлює віртуального з'єднання і використовує датаграми для відправки даних від одного комп'ютера до іншого. Це означає, що по протоколу IP пакети просто відправляються в мережу без очікування підтвердження про отримання даних (АСЬК Acknowledgment), а значить без гарантії цілісності даних. Всі необхідні дії по підтвердженню і забезпеченню цілісності даних повинні забезпечувати протоколи, що працюють на більш високому рівні.

Кожен IP-пакет містить адреси відправника і одержувача, ідентифікатор протоколу, TTL (час життя пакету) і контрольну суму для перевірки цілісності пакету. Тут також є контрольна сума, яка все ж таки дозволяє дізнатися цілісність пакету. Але про це дізнається тільки одержувач. Коли комп'ютер-одержувач приймає пакет, то він перевіряє контрольну суму тільки для себе. Якщо сума сходиться, то пакет обробляється, інакше просто відкидається. А комп'ютер-відправник пакету не зможе дізнатися про помилку, яка виникла в пакеті, і не зможе наново відправити пакет. Саме тому з'єднання по протоколу IP не можна вважати надійним.

**Зіставлення адреси ARP і RARP**.

*Протокол ARP* (Address Resolution Protocol, протокол визначення адреси) призначений для визначення апаратної (MAC) адреси комп'ютера в мережі по його IP-адресі. Перш ніж дані зможуть бути відправлені на який-небудь комп'ютер, відправник повинен знати апаратну адресу одержувача. Саме для цього і призначений ARP.

Коли комп'ютер посилає ARP запит на пошук апаратної адреси, то спочатку шукаємо цю адресу в локальному кеші. Якщо вже були звернення по даній IP-адресі, то інформація про МАС-адресу повинна зберегтися в кеші. Якщо нічого не знайдено, то в мережу посилається широкомовний запит, який одержать всі комп'ютери мережі. Вони одержать цей пакет і перевірять, якщо шукана IP-адреса належить їм або зберігається у них в кеші, то вони відповідять на запит, вказавши потрібну МАС-адресу.

Широкомовні пакети одержать всі комп'ютери локальної мережі. Жоден широкомовний пакет не пройде далі маршрутизатора. Комутатори теж дозволяють зменшити широкомовлення, тому що вони знають, до якого порту підключені комп'ютери з визначеними IP. У них знаходиться таблиця, в якій відображаються адреси підключених комп'ютерів, і якщо шуканий комп'ютер підключений до порту, то пошук полегшується. Це стосується сучасних комутаторів.

Широкомовні пакети можуть пересилатися по коаксіальному кабелю (тому що тут відсутні хаби і комутатори), а при використанні витої пари - через хаби і прості комутатори. Якщо комп'ютер розташований в іншій мережі, то його пошук відбувається небагато по іншій схемі.

*Протокол RARP* (Reverse Address Resolution Protocol, зворотний протокол визначення адреси) робить зворотне - визначає IP-адресу по відомій МАС-адресі. Процес пошуку адрес абсолютно такий же.

**Транспортні протоколи.**

На транспортному рівні є два протоколи: *UDP* і *TCP*. Обидва вони працюють поверх IP. Це означає, що коли пакет TCP або UDP опускається на рівень нижче для відправки в мережу, він потрапляє на рівень мережі прямо до протоколу IP. Тут пакету додається мережева адреса, TTL і інші атрибути протоколу IP. Після цього пакет йде далі вниз для фізичної відправки в мережу. Порожній пакет TCP не може бути відправлений до мережі, тому що він не має інформації про одержувача, ця інформація додається до пакету з IP-заголовком на рівні мережі.

***Протокол UDP*,** як і IP, для передачі даних не встановлює з'єднання з сервером. Дані просто викидаються в мережу, і протокол навіть не піклується про доставку пакету. Якщо дані на шляху до сервера зіпсуються або взагалі не дійдуть, то відправляюча сторона про це не дізнається. Отже по цьому протоколу, як і по порожньому IP, не бажано передавати дуже важливі дані.

Завдяки тому, що протокол UDP не встановлює з'єднання, він працює дуже швидко (у декілька разів швидше за TCP). Із-за високої швидкості його дуже зручно використовувати там, де дані потрібно передавати швидко і не потрібно піклуватися про їх цілісність. Прикладом можуть служити радіостанції в Інтернеті. Звукові дані просто уприсуються в глобальну мережу, і якщо слухач не одержить одного пакету, то максимум, що він відмітить - невелике заїкання в місці втрати. Але якщо врахувати, що мережеві пакети мають невеликий розмір, то заїкання буде дуже складно відмітити. Велика швидкість - великі проблеми з безпекою. Оскільки немає з'єднання між сервером і клієнтом, то немає ніякої гарантії в достовірності даних. Протокол UDP більше за інших схильний спуфінгу (spoofing, підміна адреси відправника), тому побудова на ньому захищених мереж утруднена.

Отже, UDP дуже швидкий, але його можна використовувати тільки там, де дані не мають високої цінності (можлива втрата окремих пакетів) і не мають секретності (UDP більше схильний до злому).

***Протокол TCP*** лежить на одному рівні з UDP і працює поверх IP (для відправки даних використовується IP). Саме тому протоколи TCP і IP часто об'єднують однією назвою TCP/IP, оскільки TCP нерозривно пов'язаний з IP. На відміну від UDP протокол TCP усуває недоліки свого транспорту (IP). У цьому протоколі закладені засоби встановлення зв'язку між приймачем і передавачем, забезпечення цілісності даних і гарантії їх доставки.

Коли дані відправляються в мережу по TCP, то на відправляючій стороні включається таймер. Якщо протягом певного часу приймач не підтвердить отримання даних, то буде зроблена ще одна спроба відправки даних. Якщо приймач одержить зіпсовані дані, то він повідомить про це джерелу і попросить знову відправити зіпсовані пакети. Завдяки цьому забезпечується гарантована доставка даних.

Коли потрібно відправити відразу велику порцію даних, що не вміщаються в один пакет, то вони розбиваються на декілька TCP-пакетів. Пакети відправляються порціями по декілька штук (залежить від настройок стека). Коли сервер одержує порцію пакетів, то він відновлює їх черговість і збирає дані разом (навіть якщо пакети прибули не в тому порядку, в якому вони відправлялися).

Із-за зайвих накладних витрат на установку з'єднання, підтвердження доставки і повторну пересилку зіпсованих даних протокол TCP набагато повільніше за UDP. Зате TCP можна використовувати там, де потрібна гарантія доставки і велика надійність. Хоча надійність не можна назвати сильною (немає шифрування і є можливість злому), але вона достатня і набагато більше, ніж у UDP. Принаймні, тут спуфінг не може бути реалізований так просто, як у випадку з UDP.

**Прикладні протоколи.**

*NetBIOS* (Network Basic Input Output System, базова система мережевого введення висновку) - це стандартний інтерфейс прикладного програмування. А простіше кажучи, це всього лише набір API-функцій для роботи з мережею (хоча весь NetBIOS складається тільки з однієї функції). NetBIOS був розроблений в 1983 році компанією Sytek Corporation спеціально для IBM. Система NetBIOS визначає тільки програмну частину передачі даних, тобто як повинна працювати програма для передачі даних по мережі.

Якщо подивитися на рис.1.1, то можна побачити, що NetBIOS знаходиться в самому верху схеми. Він розташований на сеансовому рівні, рівнях представлення даних і додатку. NetBIOS тільки формує дані для передачі, а фізично передаватися вони можуть тільки за допомогою іншого протоколу, наприклад TCP/IP, IPX/SPX і т.д. Це означає, що NetBIOS є незалежним від транспорту. Якщо інші варіанти протоколів верхнього рівня (що тільки формують пакети, але не передавальні) прив'язані до певного транспортного протоколу, який повинен передавати сформовані дані, то пакети NetBIOS може передавати будь-який інший протокол. Якщо написана мережева програма, працює через NetBIOS, то вона також працюватиме як в unix/windows мережах через TCP, так і в Novell-мережах через IPX.

З іншого боку, для того, щоб два комп'ютери змогли з'єднатися один з одним за допомогою NetBIOS, необхідно щоб на обох стояв хоч би один загальний транспортний протокол. Якщо один комп'ютер посилатиме NetBIOS пакети за допомогою TCP, а інший за допомогою IPX, то ці комп'ютери один одного не зрозуміють. Транспорт повинен бути однаковий. Відзначимо, що не всі варіанти транспортних протоколів за умовчанням можуть передавати по мережі пакети NetBIOS. Наприклад, IPX/SPX сам по собі цього не уміє. Щоб його навчити, потрібно мати "NWLink IPX/SPX/NetBIOS Compatible Transport Protocol". Оскільки NetBIOS найчастіше використовує як транспорт протокол TCP, який працює з установкою віртуального з'єднання між клієнтом і сервером, то по цьому протоколу можна передавати достатньо важливі дані. Цілісність і надійність передачі здійснюватиме TCP/IP, а NetBIOS дає тільки зручне середовище для роботи з пакетами і програмування мережевих додатків.

У 1985 році вже сама IBM зробила спробу перетворити NetBIOS на повноцінний протокол, який уміє не тільки формувати дані для передачі, але і фізично передавати їх по мережі. Для цього був розроблений **NetBEUI** (NetBIOS Extended User Interface, розширений призначений для користувача інтерфейс NetBIOS), який був призначений саме для опису фізичної частини передачі даних протоколу NetBIOS. Проте, NetBEUI є не маршрутізіруємим протоколом. Це означає, що якщо між двома комп'ютерами стоїть маршрутизатор і немає іншого шляху для зв'язку, то їм не вдасться встановити з'єднання через NetBEUI.

**Сокети Windows.**

Сокети (Sockets) - це програмний інтерфейс, який полегшує взаємодію між різними додатками. Сучасні сокети народилися з програмного мережевого інтерфейсу, реалізованого в ОС BSD Unix. Тоді цей інтерфейс створювався для полегшення роботи з TCP/IP, на верхньому рівні. За допомогою сокетів легко реалізувати більшість відомих протоколів, які використовуються щодня при виході в Інтернет. Достатньо тільки назвати HTTP, FTP POP3, SMTP і т.п. Всі вони використовують для відправки своїх даних або TCP, або UDP і легко програмуються за допомогою бібліотеки sockets/winsock.

**Протокол IPX/SPX**.

*Протокол IPX* (Internetwork Packet Exchange) використовується тільки в мережах фірми Novell. У вікнах є спеціальна служба Клієнт для мереж Novell, за допомогою якої можна працювати в таких мережах. IPX працює на зразок IP і UDP - без встановлення зв'язку, а значить без гарантії доставки зі всіма подальшими достоїнствами і недоліками.

*Протокол SPX* (Sequence Packet Exchange) - це транспорт для IPX, який працює зі встановленням зв'язку і забезпечує цілісність даних. Для надійності при використанні IPX, необхідно використовувати IPX/SPX або IPX/SPX11. Зараз IPX вже втрачає свою популярність.

З безлічі протоколів Інтернету більшість з них взаємозв'язана, як, наприклад, HTTP/TCP/IP. Одні протоколи можуть бути призначені для однієї мети, але абсолютно непридатні для іншої, тому що створити щось ідеальне неможливе. У кожного будуть свої достоїнства і недоліки.

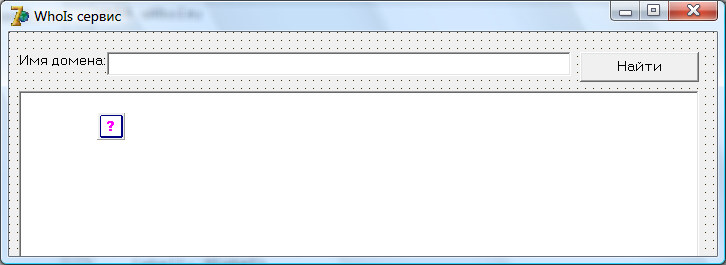
Модель OSI, прийнята ще на зорі появи Інтернету, не втратила своєї актуальності дотепер. Головне її достоїнство - приховувати складність мережевого спілкування між комп'ютерами, з чим модель OSI справляється без особливих проблем.

**Створення власної утиліти Whois**.

Сервіс Whois використовують для отримання докладної інформації про сервер, для реєстрації імені в якій-небудь зоні мережі, щоб дізнатися, чи вільний необхідний домен. Цілей використання даної утиліти достатньо багато, тому що це величезна база даних, в якій зберігається багато інформації про всі домени всесвітньої мережі.

Для створення утиліти Whois нам знадобиться бібліотека Indy. У Delphi 7 вона вже встановлена в системі, і можна приступати до її використання.

Запускайте Delphi 7. Створіть новий проект. Перенесіть на форму один компонент TEdit, одну кнопку TButton і один компонент ТМеmо (йому можна дати нове ім'я). Перейменуйте властивість Caption у кнопки на ЀЗнайтиЀ. У компонент TEdit ми вводитимемо ім'я домена, інформацію про яке хочемо одержати. Після натиснення кнопки пошуку в компоненті ТМеmо з'являтиметься все, що наша програма зможе знайти в мережі про вказаний домен.



**Рис.1.2. Зовнішній вигляд майбутньої програми**

Знайдіть закладку Indy Clients на палітрі компонентів і перенесіть на форму компонент idwhois з цієї закладки. Виділіть компонент tdwhois і перейдіть у вікно інспектора об'єктів. Подивіться на властивість Host. Тут ви повинні вказати адресу сервера, у якого є сервіс Whois. Точніше сказати, ви повинні вказати саме на цей сервіс. За умовчанням стоїть адреса whois.internic.net. Його поки міняти не треба, тому що він цілком робочий і дуже швидкий. Але якщо ви хочете змінити цю адресу, то обов'язково перевірте, який порт використовується. У принципі, настройки за умовчанням достатньо працюють для будь-яких доменів в зоні COM, ОRG і NET. Якщо вас цікавить щось специфічне, то тільки тоді вам може знадобитися зміна сервера whois. Якщо вам потрібно дізнатися інформацію про український домен .uа, то доведеться шукати український сервіс. У програмуванні компоненти Indy так само прості, як і в настройці. Створіть обробник події onClick для кнопки Знайти.

При створенні програми можна дотримуватися наступних вказівок:

* Використовувати очищення вмісту компоненту ТМеmо від тексту, що залишився від попередніх пошуків;
* При створенні рядка коду пошуку використовувати метод whois компоненту ТdWhois:
* *FindResult:=IdWhoIs.WhoIs(Edit1.Text)*
* де, *FindResult* - змінна строкового типа *Edit1* - ім'я шуканого домена.

Результат пошуку складатиметься з декількох рядків, а в змінній він виглядатиме як один довгий рядок, в якому роздільником текстових рядків є шістнадцятиричний символ #10. Щоб текст виглядав нормально, необхідно відформатувати вміст змінної *FindResult*.

Примітка. #10 - код символу переведення каретки (переходу на новий рядок), яке використовується в ОС сімейства \*nix. У Windows прийнято кінець рядка позначити парою символів #13 і #10.

Для з'ясування інформації про ім'я домена, що цікавить, необхідно запустити вашу програму, і вона сама звертається куди треба і показує вам інформацію в зручному для сприйняття вигляді. Працює все просто, тому що компонент відправляє запит серверу в Інтернеті (в даному випадку whois.internic.net) і одержує відповідь. Ніяких самостійних пошуків по базах даних не відбувається.

**Самостійна робота.**

*Завдання 1*. Згідно своєму порядковому номеру в списку групи знайти інформацію про домен:

1. versii.com

2. oligarh.net

3. yandex.ru

4. президент.рф

5. volia.com

6. news.ru

7. sport.ru

8. ukr.net

9. from-ua.com

10. mail.ru

11. i.ua

12. 1plus1.net

13. membrana.ru

14. 3dnews.ru

15. rada.gov.ua

16. musicradio.kiev.ua

17. lenta.ru

# Лабораторна робота №2.

# Сканування портів.

Кожна мережева програма при старті відкриває для себе будь-який вільний порт. Є програми, які відкривають свідомо певний порт, наприклад, для FTP це 21 порт, HTTP - 80-й порт і т.д. Якщо на сервері запущено два сервіси: FTP і WEB, то це означає, що працюють дві програми, до яких можна приєднатися по мережі. Наприклад, при приєднанні до FTP-сервера посилається запит за адресою ХХХ.ХХХ.ХХХ.ХХХ на порт 21. сервер одержує такий запит і по номеру порту визначає, що посланий запит відноситься саме до FTP-сервера, а не WEB.

Можна сказати, що мережеві порти - це щось віртуальне, що побачити неможливо, а точніше сказати - це просто число, по якому програми і ОС визначають, кому пришли дані по мережі. Якби не було портів, то комп'ютер не зміг визначити, для кого саме прийшов мережевий запит.

Сканувати порти треба для того, щоб знати, які порти відкриті, тобто які програми відкриті на видаленому комп'ютері. Так, наприклад, якщо на комп'ютері відкритий 21-й порт, то означає, на ньому працює FTP-сервер, і до нього можна спробувати приєднатися за допомогою програми FTPClient.

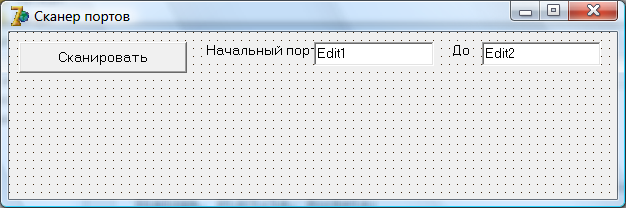
Для того, щоб зрозуміти як скануються порти, необхідно представити процес з'єднання двох комп'ютерів. Коли двоє комп'ютерів в мережі хочуть з'єднатися, то один з них посилає іншому запит з номером порту, на якому повинне відбутися з'єднання. По цьому порту інша сторона визначає, до якої програми хочуть підключитися. Якщо якийсь додаток дійсно відкрив потрібний порт і чекає з'єднання, то запрошуваний одержить відповідь про успішність спроби. Всі перевірки пароля і інші захисні механізми відбуваються вже після з'єднання з портом видаленого комп'ютера, тому можна провести з'єднання і дізнатися про доступність порту, навіть якщо програма вимагає пароль.

**Створення сканера портів**

Запустити Delphi 7. На форму перенести наступні компоненти:

* одну кнопку *Button1* (властивість Caption Сканувати.);
* два компоненти Label: *Label1* (Початковий порт) і *Label2* (До);
* два компоненти Edit: *Edit1* і *Edit2*

В результаті повинна вийти форма, зображена на рис.2.1.

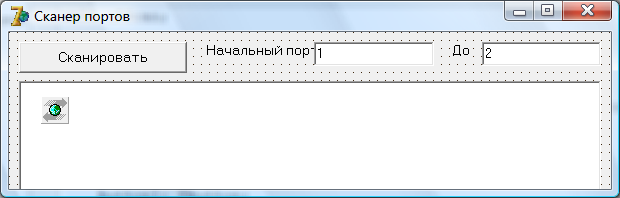


**Рис.2.1. Форма майбутнього сканера**

Після натиснення кнопки проводитиметься сканування портів, починаючи від номера, вказаного *Edit1*, до номера, вказаного *Edit2*.

Тепер переносимо на форму компонент *Client* із закладки. У ньому фірма Borland вже реалізувала всі необхідні функції для роботи з мережею по протоколу ТСР/ІР (у тому числі і сканування).

Для компонентів *Edit1* і *Edit2* у властивості *Text* встановите значення 1 і 2 відповідно. Таким чином, задамо значення за умовчанням для початкового і кінцевого порту. Перенесемо у форму ще один компонент *TMemo* і розтягнемо його на всю вільну частину форми, що залишилася. Тут відображатиметься стан сканування.

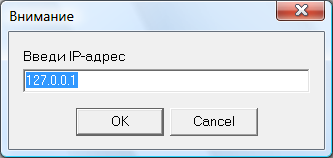


**Рис.2.2. Остаточна форма майбутнього сканера**

Створимо подію *OnClick* для кнопки **Сканувати**.

Для того, щоб сканер працював на початку програми, указуємо значення за умовчанням для адреси сканованої машини: *ipstr:=127.0.0.1* (змінна *ipstr* строкового типа). Потім за допомогою функції *InputQuery* запрошуємо у користувача ІР-адрес комп'ютера. Функція *InputQuery* виводить стандартне вікно введення (рис.2.3).

*InputQuery (‘текст заголовка вікна’, ‘текст над рядком введення’, ‘змінна типу string, куди буде записаний результат’).*



**Рис.2.3. Вікно введення ІР-адреса**

При натисненні на кнопку **ОК** запускається цикл, усередині якого перебиратимуться всі порти, номери яких вказані *Edit1* і *Edit2.*

Для того, щоб почати сканувати, треба встановити потрібний порт;

*TopClient1.RemotePort:=IntToStr(i);*

Оскільки властивість *RemotePort* є строковою, те число *i* конвертуємо в рядок. Змінна *i* указує порт, який треба сканувати. Далі відкриваємо порт:

*TopClient1.Opеn;*

Якщо відкриття порту пройшло успішно, то виводиться повідомлення, що *i-тый* порт відкритий. Для цього перевіряється властивість *TopClient1.Сonnесtеd.*

За допомогою виклику методу *Close* відкритий порт закривається. Якщо відкритий порт не закритий, то при наступній спробі відкрити відбудеться помилка.

Сканування краще проводити не більше 10 портів.

# Лабораторна робота №3

# Тема: Розробка утиліти Ping

Сервіс Ping може бути використаний для перевірки доступності видаленого сервера. **Ping** - утиліта для перевірки з'єднань в мережах на основі TCP/IP. Вона відправляє запити (ICMP Echo-Request) протоколу ICMP вказаному вузлу мережі і фіксує відповіді, що поступають (ICMP Echo-Reply). Час між відправкою запиту і отриманням відповіді (RTT, від англ. *Round Trip Time*) дозволяє визначати двосторонні затримки (RTT) по маршруту і частоту втрати пакетів, тобто побічно визначати завантаженість на каналах передачі даних і проміжних пристроях.

**ICMP** (англ. *Internet Control Message Protocol* - міжмережевий протокол повідомлень, що управляють) - мережевий протокол, що входить в стек протоколів TCP/IP. В основному ICMP використовується для передачі повідомлень про помилки і інші виняткові ситуації, що виникли при передачі даних.

Повна відсутність ICMP-відповідей може також означати, що видалений вузол (або який-небудь з проміжних маршрутизаторів) блокує ICMP Echo-Reply або ігнорує ICMP Echo-Request.

Також пінгом іноді помилкове називають час, витрачений на передачу пакету інформації в комп'ютерних мережах від клієнта до сервера і назад від сервера до клієнта. Цей час називається **лагом** (англ. *відставання; затримка, запізнювання*) або власне затримкою і вимірюється в мілісекундах. Лаг зв'язаний із швидкістю з'єднання і завантаженістю каналів на всьому протязі від клієнта до сервера.

**Практичне використання утиліти ping:**

* Визначення IP-адреси по доменному імені.
* Визначення роботи сервера. Наприклад, системний адміністратор може дізнатися чи завис тільки веб-сервер або на сервері глобальні проблеми.
* Визначення наявності зв'язку з сервером. Наприклад, проблеми з настройкою DNS-серверів на машині можна дізнатися, задавши в ping спочатку доменне ім'я, а потім IP-адресу.
* Можна дізнатися швидкість з'єднання, оскільки **ping** показує скільки запитів вдалося виконати в секунду. Також можна дізнатися якість каналу, подивившись скільки відповідей не прийшло.

Для написання власної утиліти Ping знадобиться бібліотека Internet Component Suite (ICS). Необхідно знайти її за адресою <http://www.rtfm.be/fpiette/indexuk.htm>, скопіювати файли на жорсткий диск і розархівувати їх для подальшого використання.

Запустити Delphi. При запуску буде створений новий проект. Він поки не потрібен, тому його треба закрити (File\Close All). Тепер потрібно відкрити за допомогою Delphi бібліотеку ICS. Файл, який треба відкрити, називається **IcsdelXX.dpk**, де XX - номер версії встановленого Delphi. Якщо встановлений Delphi 6 або Delphi 7, то можна відкрити **Icsdel50.dpk.** У бібліотеці немає файлів для цих версій, але 5-й встановиться без проблем.

При відкритті бібліотеки з'явиться вікно, в якому необхідно натиснути кнопку **Install**, щоб Delphi відкомпілював пакет і проїнсталліровал його. Якщо все було зроблено правильно, то повинне з'явитися вікно з переліком нових встановлених компонентів (зокрема компоненти **Ping**).

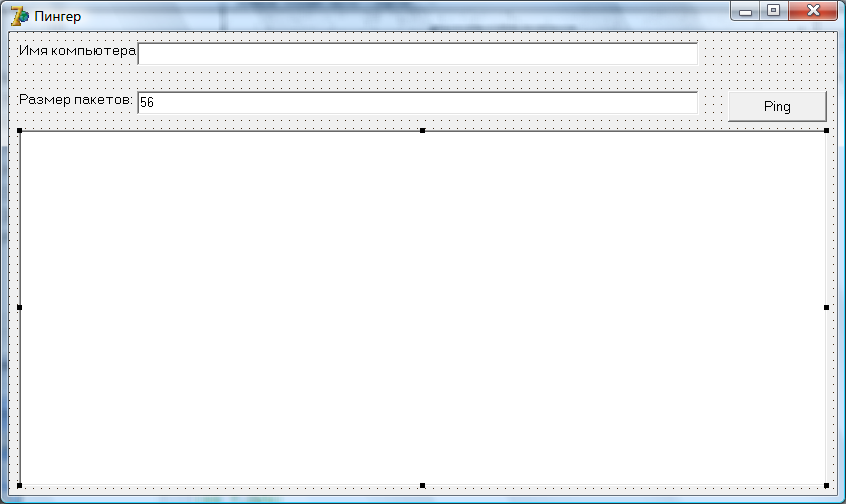
Тепер потрібно вказати Delphi, де знаходяться файли пакету, щоб він міг при компіляції проектів знайти все необхідне. Для цього треба вибрати в меню **Tools** пункт **Environment Options**. У вікні настройок Delphi, що з'явилося, треба перейти на вкладку **Library**. Клацнути на кнопці з трьома крапками напроти рядка **Library path** і внизу вікна, що з'явилося, в рядок введення ввести шлях до теки, куди був розархівований пакет. Потім натиснути кнопку **Add**. Закрити всі відкриті вікна, натискаючи численні ОК.

На палітрі компонентів з'явилася нова закладка **FPiette**.

Всі необхідні компоненти встановлені, так що тепер можна переходити до програмування. Закрити всі вікна, і на питання про збереження змін в пакеті відповідати ЀТакЀ. Створити новий проект. На форму перенести два компоненти **TLabel** і два **TEdit**. Розмістити їх так, як показано на мал. 3.1.

У **Label1** змініть властивість **Caption** на ім'я **комп'ютера**, а у **Label2** - на **Розмір пакетів**. Навпаки **Label1** повинен стояти **Edit1**. Сюди вводитиметься IP-адреса або ім'я комп'ютера, який треба пропінгувати. У **Edit2** вводитиметься розмір пакету.

Перенести на форму компонент **ТichEdit** із закладки **Win32**. У нього записуватиметься результат виконання операції. Тепер на формі треба розмістити компонент **Ping** із закладки **FPiette**, який і проводитиме пінг.



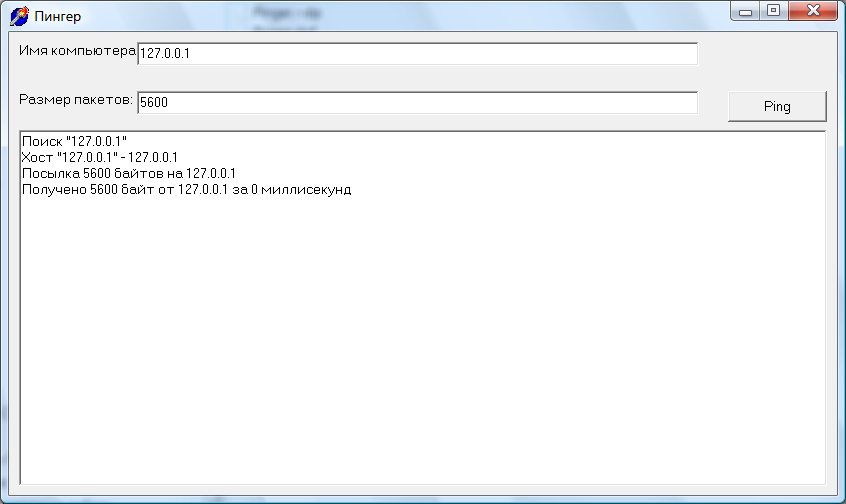
**Мал. 3.1. Форма майбутньої програми**

Створити обробник події **OnClick** кнопки. Для цього потрібне через компонент **ТichEdit** вивести повідомлення про початок пінга. Потім устанавіть розмір пакету пінга, як вказано **Edit2**. В останню чергу запустити пошук комп'ютера через DNS **(DnsLookup**). Навіть якщо ввести IP-адресу, то також відбуватиметься пошук в базі DNS.

Тепер виділивши компонент **Ping**, створити для нього обробник події **onDnsLookupDone** (коли закінчений пошук в базі DNS). У разі виявлення помилки, то вивести відповідне повідомлення Хост . не знайдений. Якщо помилок не було, то вивести в **RichEdit** результат пошуку. Потім встановити властивість **Address** компоненту **Ping** рівним адресі, знайденій в базі DNS і запустити **Ping**.

Для виведення результату пінга необхідно створити обробник події **OnEchoReply** для компоненту **Ping**. Якщо Error рівне 0, то показується повідомлення про помилку Не можу виконати операцію ping: Ѐ. Якщо ні, то показується час, за який пройшов ping. Наприклад, Одержано 5600 байт від 192.168.1.1 за 0 мілісекундЀ.

Створити обробник події **OnEchoRequest** для компоненту **ping**. Це чисто косметична поправка, яка вводить стан пінга. Подія **OnEchoRequest** відбувається тоді, коли пакет відправляється на видалену машину. Ця подія виловлюється і виводиться повідомлення про те, що зараз відправляються дані в певному розмірі на машину з вказаною адресою. Так інформація буде краще сприйматися, і з нею легше буде працювати. Наприклад, Посилка 5600 байт на 192.168.1.1.



**Мал. 3.2. Результат роботи утиліти Ping**

**Самостійна робота.**

У програму додати наступні компоненти:

* **TimeOut** - можливість зміни часу очікування відповіді на пакет;
* **TTL** - час життя пакету. Ця максимальна кількість маршрутизаторів, через які може пройти пакет. Якщо пакет проходить через більшу кількість маршрутізірующих пристроїв, чим вказано в цьому полі, то він вважається таким, що заблукав або зациклився і знищується. Це зроблено для того, щоб зациклені пакети не гуляли по мережі вічно.

# Лабораторна робота №4

## Тема: Основи створення НТTP-серверу

# Загальні теоретичні відомості. *Web-сервер*

У 1989 році Тім Бернерс-Лі запропонував інститутові CERN (Європейський центр ядерних досліджень) новий проект, який мав на меті полегшити обмін інформацією між вченими, використовуючи систему гіпертексту. В результаті виконання цього проекту Бернерс-Лі написав дві програми: браузер названий WorldWideWeb і перший веб-сервер, який працював на комп'ютері NEXTSTEP.

Мережу WWW утворюють мільйони веб-серверів, розташованих по всьому світу. Веб-сервер (англ. Web Server) — це сервер, що приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, видає їм HTTP-відповіді, зазвичай разом з HTML-сторінкою, зображенням, файлом, медіа-потоком або іншими даними. Веб-сервер — це основа Всесвітньої павутини. Для ідентифікації ресурсів (часто файлів або їх частин) в WWW використовуються ідентифікатори ресурсів URI (Uniform Resource Identifier). Для визначення місцезнаходження ресурсів в цій мережі використовуються локатори ресурсів URL (Uniform Resource Locator). Такі URL є комбінацією URI і системи DNS. Доменне ім'я (або IP-адрес) входить до складу URL для позначення комп'ютера (його мережевого інтерфейсу), на якому працює програма веб-сервер.

Веб-сервером називають як програмне забезпечення, що виконує функції веб-серверу, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює.

Клієнт, яким зазвичай є веб-браузер, передає веб-серверу запити на отримання ресурсів, позначених URL-адресами. Ресурси — це HTML-сторінки, зображення, файли, медіа-потоки або інші дані, які необхідні клієнтові. У відповідь веб-сервер передає клієнтові дані. Цей обмін відбувається по протоколу HTTP.

Пошуком (або динамічною генерацією) html-документів функції web-серверів не вичерпуються. Існують завдання, які може виконати тільки сервер, а не клієнт. Це, зокрема, ті функції, які засновані на зборі і аналізі даних, що поступають від численних клієнтів. Перерахуємо лише деякі з них:

* Зв'язок з серверними базами даних (наприклад, динамічна публікація прайс-листів, оновлення каталогів товарів, виконання запитів до БД, наприклад, динамічне формування списків, що відповідають критерію пошуку, і ін.)
* Аутентифікація користувача, обмеження доступу відповідно до прав клієнта
* Автоматичне формування і розсилка поштових повідомлень (сервери новин)
* Ведення статистики відвідин (підрахунок загального числа відвідин, числа унікальних клієнтів, частоти використання різних версій браузера, відстежування адрес, з яких користувачів потрапив на даний документ)
* Підтримка гостьових книг, форумів, чатів, лічильників відвідин і ін.

Взаємодія між клієнтом і сервером відбувається за принципом "запит-відповідь". Цей запит указується в адресному рядку браузера.

Простий запит має вигляд: *протокол://ім’я сервера.*

## 

Протокол HTTP

HTTP (HyperText Transfer Protocol - RFC 1945, RFC 2616) - протокол прикладного рівня для передачі гіпертексту.

Центральним об'єктом в HTTP є ресурс, на який указує URI в запиті клієнта. Зазвичай такими ресурсами є файли, що зберігаються на сервері. Особливістю протоколу HTTP є можливість вказати в запиті і відповіді спосіб представлення одного і того ж ресурсу по різних параметрах: формату, кодуванню, мові і так далі. Саме завдяки можливості вказівки способу кодування повідомлення клієнт і сервер можуть обмінюватися двійковими даними, хоча спочатку даний протокол призначений для передачі символьній інформації. На перший погляд це може показатися зайвою витратою ресурсів. Дійсно, дані в символьному вигляді займають більше пам'яті, повідомлення створюють додаткове навантаження на канали зв'язку, проте подібний формат має багато переваг. Повідомлення, що передаються по мережі, легкі для читання, і, проаналізувавши отримані дані, системний адміністратор може легко знайти помилку і усунути її. При необхідності роль одного з взаємодіючих застосувань може виконувати чоловік, уручну вводячи повідомлення в необхідному форматі.

На відміну від багатьох інших протоколів, HTTP є протоколом без пам'яті. Це означає, що протокол не зберігає інформацію про попередні запити клієнтів і відповіді сервера. Компоненти, що використовують HTTP, можуть самостійно здійснювати збереження інформації про стан, пов'язаної з останніми запитами і відповідями. Наприклад, клієнтський веб-додаток, що посилає запити, може відстежувати затримки відповідей, а веб-сервер може зберігати IP-адреси і заголовки запитів останніх клієнтів.

Все програмне забезпечення для роботи з протоколом HTTP розділяється на три основні категорії:

* Сервери - постачальники послуг зберігання і обробки інформації (обробка запитів).
* Клієнти - кінцеві споживачі послуг сервера (відправка запитів).
* Проксі-сервери для підтримки роботи транспортних служб.

Основними клієнтами є браузери, наприклад: Internet Explorer, Opera, Mozilla Firefox, Netscape Navigator та інші. Найбільш популярними реалізаціями веб-серверів є: Internet Information Services (IIS), Apache, lighttpd, nginx. Найбільш відомі реалізації проксі-серверів: Squid, UserGate, Multiproxy, Naviscope.

"Класична" схема HTTP-сеансу виглядає так.

* Встановлення TCP-зєднання.
* Запит клієнта.
* Відповідь сервера.
* Розривши TCP-зєднання.

Таким чином, клієнт посилає серверу запит, отримує від нього відповідь, після чого взаємодія припиняється. Зазвичай запитом клієнта є вимога передати HTML-документ або який-небудь інший ресурс, а відповідь сервера містить код цього ресурсу.

До складу HTTP-запиту, що передається клієнтом серверу, входять наступні компоненти.

Рядок стану (іноді для її позначення використовують також терміни рядок-статус, або рядок запиту).

* Поля заголовка.
* Порожній рядок.
* Тіло запиту.

Рядок стану разом з полями заголовка іноді називають також заголовком запиту.

Запит клієнта

Заголовок

Пустий рядок

Тіло запиту

Рядок стану

Поля заголовку

Метод запиту

URL ресурсу

Версія протоколу HTTP

*Мал.4.1. Структура запиту клієнта.*

**Рядок стану** має наступний формат:

метод\_запиту URL\_pecypcу версія\_протоколу\_НТТР

Розглянемо компоненти рядка стану, при цьому особливу увагу приділимо методам запиту.

**Метод**, вказаний в рядку стану, визначає спосіб дії на ресурс, URL якого заданий в тому ж рядку. Метод може приймати значення GET, POST, HEAD, PUT, DELETE і так далі Не дивлячись на велику кількість методів, для програміста по-справжньому важливі лише два з них: GET і POST.

GET. Згідно формальному визначенню, метод GET призначається для отримання ресурсу з вказаним URL. Отримавши запит GET, сервер повинен прочитати вказаний ресурс і включити код ресурсу до складу відповіді клієнтові. Ресурс, URL якого передається у складі запиту, не обов'язково повинен бути HTML-сторінку, файлом із зображенням або іншими даними. URL ресурсу може указувати на виконуваний код програми, який, при дотриманні певних умов, повинен бути запущений на сервері. В цьому випадку клієнтові повертається не код програми, а дані, що згенерували в процесі її виконання. Не дивлячись на те що, за визначенням, метод GET призначений для отримання інформації, він може застосовуватися і в інших цілях. Метод GET цілком підходить для передачі невеликих фрагментів даних на сервер.

POST. Згідно тому ж формальному визначенню, основне призначення методу POST - передача даних на сервер. Проте, подібно до методу GET, метод POST може застосовуватися по-різному і нерідко використовується для отримання інформації з сервера. Як і у випадку з методом GET, URL, заданий в рядку стану, указує на конкретний ресурс. Метод POST також може використовуватися для запуску процесу.

Методи HEAD і PUT є модифікаціями методів GET і POST.

**Версія протоколу** HTTP, як правило, задається в наступному форматі:

HTTP/версія.модифікація

**Поля заголовка**, наступні за рядком стани, дозволяють уточнювати запит, тобто передавати серверу додаткову інформацію. Поле заголовка має наступний формат:

Ім’я\_поля: Значення

Призначення поля визначається його ім'ям, яке відділяється від значення двокрапкою.

Імена деяких що найчастіше зустрічаються в запиті клієнта полів заголовка і їх призначення приведені в таблиці 1.

**Таблиця 1. Поля заголовка запиту HTTP.**

|  |  |
| --- | --- |
| **Поля заголовка HTTP -запиту** | **Значення** |
| Host | Доменне ім'я або IP-адрес вузла, до якого звертається клієнт |
| Referer | URL документа, який посилається на ресурс, вказаний в рядку стану |
| From | Адреса електронної пошти користувача, що працює з клієнтом |
| Accept | MIME-типы даних, що обробляються клієнтом. Це поле може мати декілька значень, відокремлюваних одне від іншого комами. Часто поле заголовка Accept використовується для того, щоб повідомити сервер про те, які типи графічних файлів підтримує клієнт |
| Accept-Language | Набір двосимвольних ідентифікаторів, розділених комами, які позначають мови, підтримувані клієнтом |
| Accept-Charset | Перелік підтримуваних наборів символів |
| Content-Type | MIME-тип даних, що містяться в тілі запиту (якщо запит не складається з одного заголовка) |
| Content-Length | Число символів, що містяться в тілі запиту (якщо запит не складається з одного заголовка) |
| Range | Присутній в тому випадку, якщо клієнт запрошує не весь документ, а лише його частина |
| Connection | Використовується для управління TCP-соединением. Якщо в полі міститься Close, це означає, що після обробки запиту сервер повинен закрити з'єднання. Значення Keep-Alive пропонує не закривати TCP-соединение, щоб воно могло бути використане для подальших запитів |
| User-Agent | Інформація про клієнта |

У багатьох випадках при роботі у веб-сервері тіло запиту відсутнє. При запуску CGI- сценарію для даних в запиті можуть розміщуватися в тілі запиту.

Нижче представлений приклад HTML-запроса, що генерується браузером

GET http://oak.oakland.edu/ HTTP/1.0

Connection: Keep-Alive

User-Agent: Mozilla/4.04 [en] (Win95; I)

Host: oak.oakland.edu

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png \*/\*

Accept-Language: en

Accept-Charset: iso-8859-l,\*,utf-8

Отримавши від клієнта запит, сервер повинен відповісти йому. Знання структури відповіді сервера необхідне розробникові веб-додатків, оскільки програми, які виконуються на сервері, повинні самостійно формувати відповідь клієнтові.

Подібно до запиту клієнта, відповідь сервера також складається з чотирьох перерахованих нижче компонентів.

* Рядок стану.
* Поля заголовка.
* Порожній рядок.
* Тіло відповіді.

Відповідь сервера клієнтові починається з рядка стану, який має наступний формат:

Версія\_протоколу Код\_відповіді Пояснювальне\_повідомлення

* ***Версія\_протоколу*** задається в тому ж форматі, що і в запиті клієнта, і має той же сенс.
* ***Код\_відповіді*** - це тризначне десяткове число, що представляє в закодованому вигляді результат обслуговування запиту сервером.
* ***Пояснювальне\_повідомлення*** дублює код відповіді в символьному вигляді. Це рядок символів, який не обробляється клієнтом. Вона призначена для системного адміністратора або оператора, що займається обслуговуванням системи, і є розшифровкою коди відповіді.

З трьох цифр, складових код відповіді, перша (старша) визначає клас відповіді, останні дві є номером відповіді усередині класу. Так, наприклад, якщо запит був оброблений успішно, клієнт отримує наступне повідомлення:

HТТР/1.0 200 ОК

Як видно, за версією протоколу HTTP 1.0 слідує код 200. У цьому коді символ 2 означає успішну обробку запиту клієнта, а решта двох цифри (00) — номер даного повідомлення.

У використовуваних в даний час реалізаціях протоколу HTTP перша цифра не може бути більше 5 і визначає наступні класи відповідей.

1 - спеціальний клас повідомлень, званих інформаційними. Код відповіді, що починається з 1, означає, що сервер продовжує обробку запиту. При обміні даними між HTTP-клієнтом і HTTP-сервером повідомлення цього класу використовуються достатньо рідко.

2 - успішна обробка запиту клієнта.

3 - перенаправлення запиту. Щоб запит був обслужений, необхідно зробити додаткові дії.

4 - помилка клієнта. Як правило, код відповіді, що починається з цифри 4, повертається в тому випадку, якщо в запиті клієнта зустрілася синтаксична помилка.

5 - помилка сервера. По тих або інших причинах сервер не в змозі виконати запит.

Приклади код відповідей, які клієнт може отримати від сервера, і пояснюючі повідомлення приведені в таблиці 2.

**Таблиця 2. Класи кодів відповіді серверу.**

|  |  |  |
| --- | --- | --- |
| **Код** | **Розшифровка** | **Інтерпретація** |
| 100 | Continue | Частина запиту прийнята, і сервер чекає від клієнта продовження запиту |
| 200 | OK | Запит успішно оброблений, і відповідає клієнта передаються дані, вказані в запиті |
| 201 | Created | В результаті обробки запиту був створений новий ресурс |
| 202 | Accepted | Запит прийнятий сервером, але обробка його не закінчена. Даний код відповіді не гарантує, що запит буде оброблений без помилок. |
| 206 | Partial Content | Сервер повертає частина ресурсу у відповідь на запит, що містив поле заголовка Range |
| 301 | Multiple Choice | Запит указує більш ніж на один ресурс. У тілі відповіді можуть міститися вказівки на те, як правильно ідентифікувати запрошуваний ресурс |
| 302 | Moved Permanently | Ресурс, що зажадався, більше не розташовується на сервері |
| 302 | Moved Temporarily | Ресурс, що зажадався, тимчасово змінив свою адресу |
| 400 | Bad Request | У запиті клієнта виявлена синтаксична помилка |
| 403 | Forbidden | Ресурс, що є на сервері, недоступний для даного користувача |
| 404 | Not Found | Ресурс, вказаний клієнтом, на сервері відсутній |
| 405 | Method Not Allowed | Сервер не підтримує метод, вказаний в запиті |
| 500 | Internal Server Error | Один з компонентів сервера працює некоректно |
| 501 | Not Implemented | Функціональних можливостей сервера недостатньо, щоб виконати запит клієнта |
| 503 | Service Unavailable | Служба тимчасово недоступна |
| 505 | HTTP Version not Supported | Версія HTTP, вказана в запиті, не підтримується сервером |

Відповідає використовується така ж структура полів заголовка, як і в запиті клієнта. Поля заголовка призначені для того, щоб уточнити відповідь сервера клієнтові. Опис деяких з полів, які можна зустріти в заголовку відповіді сервера, приведений в таблиці 3.

**Таблиця 3. Поля заголовка відповіді веб-сервера.**

|  |  |
| --- | --- |
| **Ім'я поля** | **Опис вмісту** |
| Server | Ім'я і номер версії сервера |
| Age | Час в секундах, що пройшов з моменту створення ресурсу |
| Allow | Список методів, допустимих для даного ресурсу |
| Content-Language | Мови, які повинен підтримувати клієнт для того, щоб коректно відобразити ресурс, що передається |
| Content-Type | MIME-тип даних, що містяться в тілі відповіді сервера |
| Content-Length | Число символів, що містяться в тілі відповіді сервера |
| Last-Modified | Дата і час останньої зміни ресурсу |
| Date | Дата і час, що визначають момент генерації відповіді |
| Expires | Дата і час, що визначають момент, після якого інформація, передана клієнтові, вважається застарілою |
| Location | У цьому полі указується реальне розташування ресурсу. Воно використовується для перенаправлення запиту |
| Cache-Control | Директиви управління кешуванням. Наприклад, no - cache означає, що дані не повинні кешуватись |

У тілі відповіді міститься код ресурсу, що передається клієнтові у відповідь на запит. Це не обов'язково повинен бути HTML-текст веб-сторінки. У складі відповіді можуть передаватися зображення, аудіо-файл, фрагмент відеоінформації, а також будь-який інший тип даних, підтримуваних клієнтом. Про те, як слід обробляти отриманий ресурс, клієнтові повідомляє вміст поля заголовка Content - type.

Нижче представлений приклад відповіді сервера на запит, приведений в попередньому розділі. У тілі відповіді міститься початковий текст HTML-документа.

HTTP/1.1 200 OK

Server: Microsoft-IIS/5.1

X-Powered-By: ASP.NET

Date: Mon, 20 OCT 2008 11:25:56 GMT

Content-Type: text/html

Accept-Ranges: bytes

Last-Modified: Sat, 18 Oct 2008 15:05:44 GMT

ETag: "b66a667f948c92:8a5"

Content-Length: 426

<html>

<body>

<form action='http://localhost/Scripts/test.pl'>

<p>Operand1: <input type='text' name='A'></p>

<p>Operand2: <input type='text' name='B'></p>

<p>Operation:<br>

<select name='op'>

<option value='+'>+</option>

<option value='-'>-</option>

<option value='\*'>\*</option>

<option value='/'>/</option>

<select></p>

<input type='submit' value='Calculate!'>

</from>

</body>

</html>

Поля заголовка і тіло повідомлення можуть бути відсутніми, але рядок стану є обов'язковим елементом, оскільки указує на тип запиту/відповіді.

Поле з ім'ям Content-type може зустрічатися як в запиті клієнта, так і відповідає сервера. Як значення цього поля указується MIME -тип вмісту запиту або відповіді. MIME -тип також передається в полі заголовка Accept, присутнього в запиті.

Специфікація MIME (Multipurpose Internet Mail Extension — багатоцільове поштове розширення Internet) спочатку була розроблена для того, щоб забезпечити передачу різних форматів даних у складі електронних листів. Проте застосування MIME не вичерпується електронною поштою. Засоби MIME успішно використовуються в WWW і, по суті, стали невід'ємною частиною цієї системи.

Стандарт MIME розроблений як розширювана специфікація, в якій мається на увазі, що число типів даних буде рости у міру розвитку форм представлення даних. Кожен новий тип в обов'язковому порядку повинен бути зареєстрований в IANA (Internet Assigned Numbers Authority).

До появи MIME комп'ютери, що взаємодіють по протоколу HTTP, обмінювалися виключно текстовою інформацією. Для передачі зображень, як і для передачі будь-яких інших двійкових файлів, доводилося користуватися протоколом FTP.

Відповідно до специфікації MIME, для опису формату даних використовуються тип і підтип. Тип визначає, до якого класу відноситься формат вмісту HTTP-запиту або HTTP-відповіді. Підтип уточнює формат. Тип і підтип відділяються один від одного косою межею:

*тип/підтип*

Оскільки в переважній більшості випадків у відповідь на запит клієнта сервер повертає початковий текст HTML-документа, то в полі Content-type відповіді зазвичай міститься значення text/html. Тут ідентифікатор text описує тип, повідомляючи, що клієнтові передається символьна інформація, а ідентифікатор html описує підтип, тобто указує на те, що послідовність символів, що міститься в тілі відповіді, є описом документа на мові HTML.

Перелік типів і підтипів MIME достатньо великий. У таблиці 4 приведені приклади MIME -типів, що найчастіше зустрічаються в заголовках HTML-запитів і відповідей.

**Таблиця 4. MIME типи даних.**

|  |  |  |
| --- | --- | --- |
| **Тип/підтип** | **Розширення** | **Опис** |
| application/pdf | .pdf | Документ, призначений для обробки Acrobat Reader |
| application/msexcel | .xls | Документ у форматі Microsoft Excel |
| application/postscript | .ps, .eps | Документ у форматі PostScript |
| application/x-tex | .tex | Документ у форматі ТИХ |
| application/msword | .doc | Документ у форматі Microsoft Word |
| application/rtf | .rtf | Документ у форматі RTF, що відображається за допомогою Microsoft Word |
| image/gif | .gif | Зображення у форматі GIF |
| image/ jpeg | .jpeg, .jpg, | Зображення у форматі JPEG |
| image/tiff | .tiff, .tif | Зображення у форматі TIFF |
| image/x-xbitmap | .xbm | Зображення у форматі XBitmap |
| text/plain | .txt | ASCII-текст |
| text/html | . html . htm | Документ у форматі HTML |
| audio/midi | .midi, .mid | Аудіофайл у форматі MIDI |
| audio/x-wav | .wav | Аудіофайл у форматі WAV |
| message/rfc822 |  | Поштове повідомлення |
|  |  |  |
| message/news |  | Повідомлення в групи новин |
| video /mpeg | .mpeg, .mpg, .mpe | Відеофрагмент у форматі MPEG |
| video/avi | .avi | Відеофрагмент у форматі AVI |

Для однозначної ідентифікації ресурсів в мережі Веб-сервера використовуються унікальні ідентифікатори URL.

Одноманітний ідентифікатор ресурсу URI (Uniform Resource Identifier) є короткою послідовністю символів, що ідентифікує абстрактний або фізичний ресурс. URI не указує на те, як отримати ресурс, а тільки ідентифікує його. Це дає можливість описувати за допомогою RDF (Resource Description Framework) ресурси, які не можуть бути отримані через інтернет (імена, назви і тому подібне). Найвідоміші приклади URI - це URL і URN.

URL (Uniform Resource Locator) - це URI, який, крім ідентифікації ресурсу, надає ще і інформацію про місцезнаходження цього ресурсу.

URN (Uniform Resource Name) - це URI, який ідентифікує ресурс в певному просторі імен, але, на відміну від URL, URN не указує на місцезнаходження цього ресурсу.

URL має наступну структуру:

<схема>://<логин>:<пароль>@<хост>:<порт>/<URL-путь>

де:

* схема - схема звернення до ресурсу (зазвичай мережевий протокол);
* логін - ім'я користувача, використовуване для доступу до ресурсу;
* пароль - пароль, що асоціюється з вказаним ім'ям користувача;
* хост - повністю прописане доменне ім'я хоста в системі DNS або IP-адрес хоста;
* порт - порт хоста для підключення;
* URL-шлях - уточнююча інформація про місце знаходження ресурсу.

Загальноприйняті схеми (протоколи) URL включають протоколи: ftp, http, https, telnet, а також:

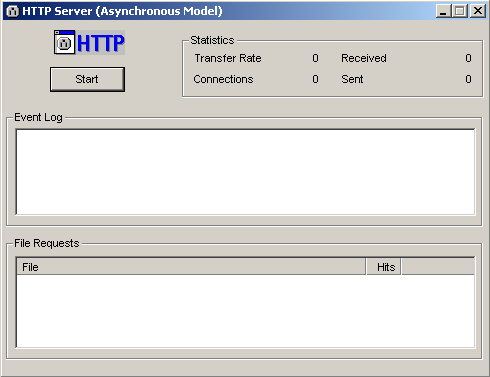
* gopher — протокол Gopher;
* mailto — адреса електронної пошти;
* news — новини Usenet;
* nntp — новини Usenet через протокол NNTP;
* irc — протокол IRC;
* prospero — служба каталогів Prospero Directory Service;
* wais — база даних системи WAIS;
* xmpp — протокол XMPP (частина Jabber);
* file — ім'я локального файлу;
* data — безпосередні дані (Data: URL);

# Практична частина

Основна форма програми зображена на рис.2.1.

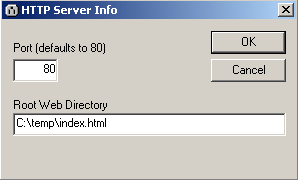
Поле Event Log являє собою журнал подій. Вньому відображається інформація, щодо подій , які відбуваються з сервером.

В полі File Requests виводиться інформація про запити.



мал.4.2. Вигляд форми основного вікна

При натисканні кнопки Start відкривається вікно, в якому потрібно ввести номер порту і кореневий каталог серверу. Це вікно зображене на рис.2.2.



мал.4.3. Форма вікна вводу інформації

Для проведення перевірки працездатності створемо html-файл і помістимо його в каталог C\temp. Файл матиме наступний зміст:

<html>

<head>

<title>Testing</title>

</head>

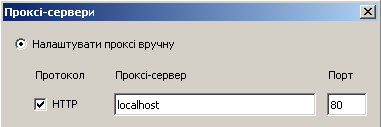
<body>

<h1>This is the test</h1>

</body>

</html>

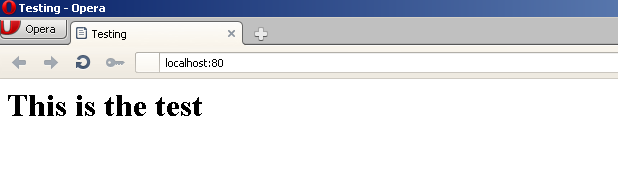
Також потрібно налаштувати браузер, а саме, в налаштуваннях проксі-серверу введемо параметри, що зображені на наступному рисунку:



мал.4.4. Параметри налаштування браузеру.

Наступним кроком вводимо в поле адреси браузера адресу сервера: <http://localhost:80/>.

Отриманий результат тестування зображений на мал. 4.5.



мал.4.5.

# Лабораторна робота №5

Тема: Отримання інформації о сокетах.

# Сокети (Sockets) - всього лише програмний інтерфейс, який полегшує взаємодію між різними додатками. Сучасні сокети народилися з програмного мережевого інтерфейсу, реалізованого в ОС BSD Unix. Тоді цей інтерфейс створювався для полегшення роботи з TCP/IP, на верхньому рівні.

# За допомогою сокетів легко реалізувати більшість відомих вам протоколів, які використовуються щодня при виході в Інтернет. Достатньо тільки назвати HTTP, FTP POP3, SMTP і далі в тому ж дусі. Всі вони використовують для відправки своїх даних або TCP, або UDP і легко програмуються за допомогою бібліотеки sockets/winsock.

Основні функції WinSock

# Бібліотека WinSock складається з одного лише файлу Winsock.dll. Вона дуже добре підходить для створення простих додатків, тому що в ній реалізовано все необхідне для створення з'єднання і прийому/передачі файлів. Зате сніфер створювати навіть не намагайтеся. У WinSock немає нічого для доступу до заголовків пакетів. MS обіцяла вбудувати ці необхідні просунутому програмісту речі WinSock2, але як завжди все закінчилося тільки словами.

# Чим хороша ця бібліотека, так це тим, що всі її функції однакові для багатьох платформ і мов програмування. Так, наприклад, коли ми напишемо сканер портів, його легко можна буде перенести на мову C/C++ і навіть написати щось подібне \*nix, тому що там мережеві функції називаються так само і мають практично ті ж параметри. Різниця між мережевою бібліотекою Windows і Linux мінімальна, хоч і є. Але так і повинно бути, адже Microsoft не може по-людськи, і їх програмістам обов'язково треба виділитися.

# Ми вивчатимемо WinSock2, а Delphi підтримує тільки першу версію. Щоб вона змогла побачити другу, потрібно підключити заголовні файли для цієї версії.

# Вся робота мережевої бібліотеки побудована навколо поняття socket - це як би віртуальний мережевий канал. Для з'єднання з сервером ви повинні підготувати такий канал до роботи і потім можете з'єднуватися з будь-яким портом сервера.

Загальний алгоритм роботи з сокетами.

# Ініціалізували бібліотеку WinSock.

# Ініціалізували socket (канал для зв'язку). Після ініціалізації у нас повинна бути змінна, вказуюча на новий канал. Створений сокет - це, можна сказати, відкритий порт на вашому комп'ютері. Порти є не тільки на сервері, але і у клієнта, і коли відбувається передача даних між комп'ютерами, то вона відбувається між мережевими портами.

# Можна приєднуватися до сервера. У кожній функції для роботи з мережею першим параметром обов'язково указується змінна, вказуюча на створений канал, через який відбуватиметься з'єднання.

Ініціалізація WinSock

# Найперше, що треба зробити - ініціалізувати бібліотеку (для UNIX-подібних ОС це не потрібно робити). Для цього необхідно викликати функцію wsAStartup. У неї є два параметри:

# Версія WinSock, яку ми хочемо використовувати. Для версії 1.0 потрібно вказати MAKEWORD(I,O), АЛЕ нам потрібна друга, означає, указуватимемо MAKEWORD(2,0).

# Структура типу TWSADATA, В якій буде повернена інформація про знайдене WinSock.

# Тепер дізнаємося, як потрібно закривати бібліотеку. Для цього потрібно викликати функцію wsACieanup, у якої немає параметрів. У принципі, якщо ти не закриєш WinSock, то нічого критичного не відбудеться. Після виходу з програми все саме закриється, просто звільнення непотрібного відразу після використання є хорошим тоном в програмуванні. Приклад ініціалізації. Давайте відразу напишемо приклад, який ініціалізує WinSock і виводити на екран інформацію про нього. Створіть в Delphi новий проект. Тепер до нього треба підключити заголовні файли WinSock другої версії. Для цього треба перейти в розділ uses і додати туди модуль winsock2.

# Якщо ви спробуєте зараз скомпілювати цей порожній проект, то Delphi лаятиметься на доданий модуль. Це тому, що вона не може знайти самі файли. Тут можна поступити декількома способами.

Підключення заголовних файлів

# Збережіть новий проект в яку-небудь директорію і туди ж скопіюйте файли winSock2.pas, ws2tcpip.inc, wsipx.inc, wsnwlink.inc иwsnetbs.inc. Незручність цього способу - в кожен проект, що використовує Winsock2, треба закидати заголовні файли.

# Можна помістити ці файли в теку Delphi\Lib, і тоді вже точно будь-який проект знайде їх.

# Можна покласти файли в окрему директорію, а потім підключити її до Delphi.

# Щоб підключити директорію із заголовними файлами, в Delphi потрібно вибрати в меню Tools пункт Environment options. У вікні, що з'явилося, потрібно перейти на вкладку Library. Тут відбувається настройка шляхів, з якими працює середовище розробки Delphi. Нас цікавить перший рядок Library Path. У цьому рядку перераховані шляхи до директорій із заголовними файлами і початковими кодами компонентів. Ось сюди і потрібно додати шлях до директорії з файлами WinSock2. Для цього є два способи:

# 1. Додати ручками в кінець рядка крапку з комою ";" і після цього написати повний шлях до файлів.

# 2. Клацнути на кнопці з трьома крапками праворуч від рядка введення, і перед вами відкриється вікно.

# Тут зверху знаходиться список директорій і під ним рядок введення для нового шляху. Праворуч від рядка введення є кнопка з трьома крапками. Якщо клацнути на цій кнопці, то ви побачите стандартне вікно вибору теки. Знайдіть потрібну теку із заголовними файлами і натисніть ОК. Тепер в рядку введення повинен відображатися повний шлях до потрібної директорії. Щоб додати його потрібно клацнути на кнопці Add. Все, шлях доданий, і можна закривати всі вікна натисненням кнопок ОК.

Отримання інформації про сокети

# Ось тепер спробуємо ініціалізувати бібліотеку WinSock. Для цього перенесіть на створену нами форму три рядки введення і кнопку.

# Після цього створіть обробник події onclick для кнопки і напишіть там наступний текст:

# procedure Т. Forml.ButtonlClickf Sender: TObject);info:TWSADATA;

# begin

# WSAStartup(MAKEWORD(2,0), info);

# VersionEdit.Text:=IntToStr(info.wVersion);

# DescriptionEdit.Text:=info.szDescription;

# SystemStatusEdit.Text:=infо.szSystemStatus;

# WSACleanup;

# end;

# В самому початку запускається WinSock за допомогою виклику функції wsAStartup. У ньому запрошується друга версія, а інформація про поточний стан буде повернена в структуру info. Після цього виводимо одержану інформацію із структури в головне вікно програми.

Підготовка роз'єму

# Перш ніж проводити з'єднання з сервером, треба ще підготувати socket до роботи. Цим і займемося. Для підготовки потрібно виконати функцію socket, у якої є три параметри.

# Тип використовуваної адресації. Нас цікавить Інтернет, тому ми указуватимемо PF\_INET АБО AF\_INET. Як бачите, обидва значення дуже схожі і показують одну і ту ж адресацію. Перший з них ми використовуватимемо при синхронній роботі, а другий - при асинхронній.

# Завжди краще відразу ж визначитися, з яким типом порту ми зараз працюємо. Базовий протокол. Тут ми повинні вказати, на основі якого протоколу відбуватиметься робота. Якщо ви прочитали документи про мережі те повинні знати, що існує два базових протоколу: TCP (з надійним з'єднанням) і UDP (що не проводить з'єднань, а що просто подає дані в порт). Для TCP в цьому параметрі треба вказати SOCKSTREAM, а якщо потрібен UDP, то указуйте SOCK\_DGRAM. Ось тут ми можемо указувати, який конкретно протокол нас цікавить. Можливих значень тут дуже багато (наприклад IPPROTO\_IP, IPPORT\_ECHO, IPPORT\_FTP і т. д.). Якщо хочете побачити все, то відкривайте файл winsock2.pas і запускайте пошук по IPPORT\_, І все, що ви знайдете, - це і будуть можливі протоколи.

Синхронність/асинхронність

# Тепер познайомимося з синхронністю і асинхронністю роботи порту. Різниця в цих двох режимах наступна. Синхронна робота: коли ви викликаєте функцію, то програма зупиняється і чекає повного її виконання. Допустимо, що ви запитали з'єднання з сервером. Програма чекатиме, поки не відбудеться реальне з'єднання або помилка.

# Асинхронна робота: у цьому режимі програма не спотикається об кожну мережеву функцію. Допустимо, що ви зробили все той же запит на з'єднання з сервером. Ваша програма посилає запит на з'єднання і тут же продовжує виконувати наступні дії, не чекаючи фізичного контакту з сервером. Це дуже зручно (але важко в програмуванні), тому що можна використовувати час очікування контакту в своїх цілях. Єдине, що ви не можете робити - викликати мережеві функції, поки не відбудеться реального фізичного з'єднання. Недолік в тому, що самому програмісту доводиться стежити за тим, коли закінчиться виконання функції і можна буде далі працювати з мережею.

З'єднання

# Сокет готовий, а значить можна провести з'єднання з сервером. Для цього в бібліотеки WinSock є функція connect. У цієї функції є три параметри:

# Змінна-сокет, яку ми одержали після виклику функції socket.

# Структура Типу TSockAddr.

# Розмір структури, вказаної в другому параметрі. Для того, щоб дізнатися розмір, можна скористатися функцією sizeOf і вказати як параметр структуру.

# Структура TSockAddr дуже складна, і описувати її повністю немає сенсу.

# Краще ми познайомимося з нею на практиці, перераховуємо тільки основні поля, які повинні бути заповнені:

# •sinfamily - сімейство використовуваної адресації. Тут потрібно указувати те ж, що указували в першому параметрі при створенні сокета (для нас це PF\_INET або AF\_INET);

# •sinaddr - адреса сервера, куди ми хочемо приєднатися;

# •sinjport - порт, до якого ми хочемо підключитися.

# На ділі це виглядатиме так:

# var

# addr: TSockAddr;

# begin

# addr.sin\_family :\*= AF^INET;

# addr.sin\_addr := ServerName;

# addr.sin\_port := htons(21);

# connect(FSocket @addr, sizeof(addr));

# end;

# Ну і наприкінці - функція для закриття з'єднання - closesocket. Як параметр потрібно вказати змінну - сокет.

Проектування поштового клієнта.

# Лабораторна робота №6

Тема: Проектування поштового клієнта.

E-mail клієнт - програма, якою ми користуємося регулярно. Кількість ЀмильницьЀ, пропонованих розробниками росте не днями, а годинами. Вибір великий, але часто всі ці багатофункціональні монстри забезпечені тими фішками, які можуть взагалі ніколи не потрібно середньостатистичному користувачу, але за них доводитися платити.

Щоб заощадити (а може і запрацювати) бабло ми покажемо як написати свій власний e-mail клієнт, який умітиме відправляти і приймати пошту. Ми розуміємо, що закодіть таку програмку, використовуючи компоненти якось не по-хацкерські, тому ми ускладнюємо завдання і розглянемо поштовик на WinSock API. Одержані знання тобі стануть в нагоді не тільки для створення ЀпоштовиківЀ, але і інших корисних хацкеру мережевих тулз.

Поштові протоколи

Перш ніж приступити до практики і терзання клавіатури, давай розберемося з теорією прийому і передачі поштових повідомлень. Ти напевно знаєш (а якщо не знаєш, то сором тобі і ганьба), що для передачі електронних листів існує старий перевірений протокол smtp (simple mail transfer protocol - простий протокол передачі електронної пошти), а для прийому найбільшою популярністю користується POP3.Спецификация цих протоколів описана в RFC 2821 і RFC 1225. Я відразу ж рекомендую тобі їх викачати, оскільки в цих доках описано багато цікавих речей, про які через обмеження статті я розповісти не смогу.

SMTP

Коли в своєму поштовому клієнті ти сформував лист і натиснув кнопочку ЀвідправитиЀ, твоя мильниця починає з'єднуватися з вказаним в настройках smtp сервером для передачі спеціальних команд, за допомогою яких і буде відправлене твоє повідомлення. Давай розглянемо це процес на прикладі ЀручноїЀ відправки листа. За паперовим конвертом і марками можеш не бігти, все що нам буде потрібно для цього - telnet.exe, який входить в постачання windows.

Запускай cmd.exe і набирайте команду: telnet your smtp server порт. Порт smtp сервера за умовчанням - 25, але деякі адміни його змінюють. Після установки з'єднання з видаленим сервером ми одержимо вітання:

Коди відповідей SMTP сервера

Ти відмітив, що майже на кожну нашу команду сервер відповідає відповідним цифровим кодом. Завдяки цим відповідям, ми можемо аналізувати ситуацію, що відбувається. Наприклад, якщо ми відправимо некоректну команду, то сервер поверне нам код 500, який означає ЀUnrecognized command (команда не розпізнана)Ѐ. Щоб всі листи передавалися без сучка і задирочки, в програмі клієнті треба реалізувати перевірку повертаних сервером команд, а для цього в свою чергу необхідно знати їх значення. Всі можливі коди (на підставі RFC 2821) я привів в табличці нижче.

211 - Стан системи

214 - Інформація про команди підтримуваних сервером

220 - Вітання, сервер готовий до отримання і виконання команд

221 - Закриття каналу для передачі даних. Розрив з'єднання

250 - Операція успішно виконана

251 - Користувач не є локальним. Пошта пересилатиметься по прямому шляху

252 - Сервер не може перевірити вказаний ящик, але повідомлення поставлене в чергу і буде спроба його доставки

354 - Дозвіл передачі даних повідомлення. По завершенню потрібно відправити .

421 - Для даного домену продовження неможливе, сесія закривається

450 - Виконання операції неможливе, можливо поштова скринька зайнята

451 - Виконання операції перерване із-за помилки

452 - Операція не виконана з браку місця.

500 - Синтаксична помилка або невідома команда

501 - Синтаксична помилка в параметрах або аргументах команди

502 - Команда не підтримується. Сервер не підтримує дану команду.

550 - Операція не можлива

552 - Операція перервана із-за перевищення дискової квоти

553 - Операція не виконана, оскільки ім'я ящик є неприпустимим

Для роботи з мережею, в Delphi використовуються Indy компоненти, які ви можете знайти на декількох вкладках середовища розробки. Знайдіть на вкладці Indy Clients - компонент IdPOP3, на вкладці Indy Misc - компонент IdMessage. За допомогою компоненту IdPOP3 ми зможемо здійснити підключення до поштового сервера по протоколу POP3 і одержати необхідне нам к-ть повідомлень. Компонент IdMessage використовуватиметься, як буфер для одержуваного листа. Ще нам буде потрібно memo для відображення тексту листа, і кнопка для завантаження чергового повідомлення.

Отримання одного листа:

procedure TForm1.Button1Click(Sender: TObject);

begin

POP3.Host:=’mail.58r.ru’; // адреса поштового сервера

POP3.Port:=110; // порт по якому здійснюватиметься підключення

POP3.Username:=’test+58r.ru’; // Логін користувача

POP3.Password:=’12345666\_; // пароль користувача

IdMessage.Clear; // очищення буфера для повідомлення

POP3.Connect; // підключення по протоколу POP3, по настройках в компоненті IdPOP3, з ім'ям POP3

Memo1.Clear; // очищення компоненту memo для відображення тексту листа.

POP3.Retrieve(1,IdMessage); // отримання одного повідомлення

Memo1.Lines.AddStrings(IdMessage.Body); // передача повідомлення з компоненту IdMessage в memo

POP3.Delete(1); // видалення з сервера, одержаного повідомлення

POP3.Disconnect; // розрив зв'язку

end; Звичайно потрібно завантажити декілька повідомлень, все що є на сервері. Я вирішив обмежитися 10 листами, щоб не ускладнювати код. Одержувані повідомлення зберігаються в тій же теці що і поштова програма.

Отримання 10 листів:

procedure TForm1.Button2Click(Sender: TObject);

label f;

var

mailcicl:integer; // лічильник одержуваних листів

addr:string; // тека в якій зберігатимуться одержувані листи

begin

POP3.Host:=’mail.58r.ru’; // адреса поштового сервера

POP3.Port:=110; // порт по якому здійснюватиметься підключення

POP3.Username:=’test+58r.ru’; // Логін користувача

POP3.Password:=’12345666\_; // пароль користувача

addr:=ExtractFilePath(Application.ExeName); // отримання адреси теки, в якій знаходитися наша програма

POP3.Connect; // підключення до сервера

for mailcicl:=1 to 10 do // цикл отримання листів

begin

if POP3.CheckMessages<1 then goto f ; // Якщо листів немає, то вийти і циклу

IdMessage.Clear; // очищення буфера для повідомлення

Memo1.Clear; // очищення компоненту memo для відображення тексту листа.

POP3.Retrieve(1,IdMessage); // отримання одного повідомлення

Memo1.Lines.AddStrings(IdMessage.Body); // передача повідомлення з компоненту IdMessage в memo

Memo1.Lines.SaveToFile(addr+inttostr(mailcicl)+’.txt’); // збереження листа в теці, де знаходиться наша програма

POP3.Delete(mailcicl); // видалення листа на сервері

end;

f: // мітка виходу з циклу

POP3.Disconnect; // розрив з'єднання з сервером end;

Кожне повідомлення має атрибути - це майл відправника, тема повідомлення, важливість і інші елементи. Вивчіть компонент IdMessage, щоб знати всі можливі атрибути.

Отримання листа з атрибутам:

procedure TForm1.Button3Click(Sender: TObject);

var

i,numPosts: Integer;

begin

POP3.Host:=’mail.58r.ru’; // адреса поштового сервера

POP3.Port:=110; // порт по якому здійснюватиметься підключення

POP3.Username:=’test+58r.ru’; // логін користувача

POP3.Password:=’12345666 ; // пароль користувача

IdMessage.Clear; // очищення буфера для повідомлення

POP3.Connect; // підключення по протоколу POP3, по настройках в компоненті IdPOP3, з ім'ям POP3

Memo1.Clear; // очищення компоненту memo для відображення тексту листа.

POP3.Retrieve(1,IdMessage); // отримання одного повідомлення

Memo1.Lines.AddStrings(IdMessage.Body); // передача повідомлення з компоненту IdMessage в memo

// висновок в компоненти Label інформації про повідомлення.

Label1.Caption := IdMessage.From.Text;

Label2.Caption := IdMessage.Recipients.EmailAddresses;

Label3.Caption := IdMessage.CCList.EMailAddresses;

Label4.Caption := IdMessage.Subject;

Label5.Caption := FormatDateTime(’dd mmm yyyy hh:mm:ss’, IdMessage.Date);

Label6.Caption := IdMessage.ReceiptRecipient.Text;

Label7.Caption := IdMessage.Organization;

POP3.Delete(1); // видалення з сервера, одержаного повідомлення

POP3.Disconnect; // розрив зв'язку

end;

Відправка пошти здійснюється по протоколу SMTP, компонент IdSMTP, ви зможете знайти його на вкладці Indy Clients. У даному прикладі відправка пошти здійснюватиметься після авторизації сервері, лист відправляється з майла test@58r.ru, на майл test@58r.ru .

Відправка одного листа:

procedure TForm1.Button4Click(Sender: TObject);

begin

SMTP.Host:=’mail.58r.ru’; // адреса поштового сервера

SMTP.Port:=25; // порт по якому здійснюватиметься підключення

SMTP.Username:=’test+58r.ru’; // логін користувача

SMTP.Password:=’12345666'; // пароль користувача

SMTP.AuthenticationType:=atLogin; // тип підключення до сервера - з авторизацією

with IdMessage do

begin

Body.Assign(Memo1.Lines); // передача даних з memo в компонент IdMessage, для подальшої відправки

From.Text := ‘test@58r.ru’; // майл відправника листа

Recipients.EMailAddresses := ‘test@58r.ru’; // майл одержувача листа

Subject := ‘Programmersclub.ru’; // тема листа

end;

SMTP.Connect; // з'єднання з поштовим сервером

try

showmessage(’Відбулося підключення до сервера’); // повідомлення про вдале підключення до сервера

SMTP.Send(IdMessage); // відправка листа на сервер

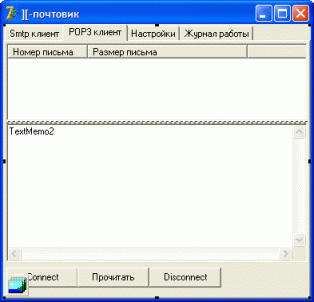
finally

SMTP.Disconnect; // розрив з'єднання

end; end;

Delphi ісходникі поштового клієнта:

Коли ви відкриєте ісходникі, ви побачите 4 кнопки, кожному даному варіанту відповідає кнопка на формі, в memo ви зможете бачити текст листа, внизу форми розміщено декілька label, в яких відображаються атрибути листа. Для роботи з мережею використовуються три Indy компоненти IdPOP3, IdSMTP, IdMessage.



мал.5.1.

# Лабораторна робота №7

## Тема: Розробка сканеру локальної мережі.

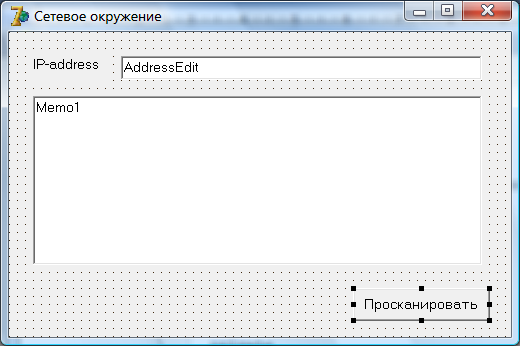
Расшарені ресурси - це будь-які ресурси комп'ютера (директорії, диски або принтери), до яких відкритий вільний доступ з мережі. Якщо комп'ютер підключений до локальної мережі, то для обміну файлами найчастіше роблять доступним (расшаривають) який-небудь диск або теку. А якщо комп'ютер має вихід в Інтернет, то до цих ресурсів можна пробратися з будь-якої точки Землі, якщо не вжиті заходи обережності.

Що дуже багато починають користувачів, знаходячись в мережі, мають расшарені ресурси, не захищені паролем. Зараз таких користувачів стає вже набагато меншим, але таке диво можна ще зустріти практично у будь-якого крупного провайдера.

Як можна здогадатися, у будь-якого провайдера є купа IP-адрес, і перебирати їх уручну достатньо складна справа. Щоб автоматизувати процес пошуку, використовують спеціальні сканери расшарених ресурсів. Простий варіант такого сканера можна написати.

Запустити Delphi і на формі встановлюємо один компонент **TEdit** (у властивості **Name** вказати **AddressEdit**) і один **TMemo**(тут у властивості **Name** залишимо значення за умовчанням **Memo1**). Компоненти потрібно належним чином оформити і додати кнопку **Просканувати**. На мал. 4.1 продемонстрований варіант форми.

У компонент **AddressEdit** користувачем вводитиметься адреса сканованого комп'ютера. В даному випадку обмежимося скануванням тільки однієї адреси. У компоненті відображатимуться знайдені ресурси, лежачі у вільному доступі.

****

Мал. 6.1 Форма майбутньої програми.

Тепер нам потрібно створити обробник події OnClick кнопки і написати програму.

При створенні програми на самому початку відбувається заповнення структури NetcontainerOpen, яка оголошена в розділі var як що належить типу NETRESOURCE.

У неї потрібно заповнити наступні п'ять полів.

**dwScope** - в цьому параметрі потрібно вказати рамки перераховуваних ресурсів. Щоб пошук відбувався в мережі треба вказати RESOURCE\_GLOBALNET.

**dwType** - тут указується тип перераховуваних ресурсів. Можна вказати RESOURCETYPE\_DISK для дисків RESQURCETYPE\_PRINT для принтерів і RESGURCETYPE\_ANY для всього підряд.

**ipLocaiName** - цей параметр потрібно обнулити.

**ipRemoteName** - тут потрібно вказати NetBIOS-ім'я сканованого комп'ютера або IP-адресу. Якщо ви указуєте адресу, то спочатку потрібно додати два слеши \.

**ipProvider** - ім'я власника ресурсу. Якщо воно не відоме, то потрібно вказати **nil**.

Після заповнення структури потрібно відкрити процес сканування. Для цього існує функція **WNetopenEnum** з наступними п'ятьма параметрами.

* Область сканування. Тут знову указуємо RESOURCE\_GLOBALNET.
* Тип сканованих ресурсів. Знову указуємо все підряд - RESOURCETYPE\_\_ANY.
* Тут потрібно вказати, які ресурси треба перераховувати. Якщо потрібне все підряд, то просто вкажіть 0. Інші можливі значення:
* RESOURCEUSAGE\_CONNECTABLE - що підключаються, і RESOURCEUSAGE\_CONTAINER - що зберігаються.
* Заповнена структура.
* Змінна типу **THandle**, яка використовуватиметься надалі.

Після того, як відкрито перелік, можна приступати до його реалізації. Для цього запускається нескінченний цикл:

*while TRUE do*

*begin*

*end;*

Усередині циклу постійно викликається функція **WNetEnumResource**. Якщо вона повертає помилку (результат не рівний NO\_ERROR), то перелік закривається за допомогою **WNetCloseEnum**, і відбувається вихід з процедури, тому що більше відкритих ресурсів немає. У функції **WNetEnumResource** є чотири параметри:

1. Тут потрібно вказати ту ж змінну, яка була вказана в останньому параметрі при відкритті переліку **WNetOpenEnum**.
2. Тут потрібно вказати змінну, в якій зберігається число необхідних до повернення ресурсів наприклад, число 2000). Після того, як функція виконається, в цій змінній буде не 2000, а кількість реально відкритих ресурсів.
3. Тут повинен бути масив структур **TNetResource**. Його довжина повинна бути достатній для зберігання поверненій інформації про відкриті ресурси. Якщо запрошується максимум 2000 ресурсів, означає, масив повинен складатися з 2000 структур (ResourceBuffer: **array[1..2000] of TNetResource;)**.
4. Розмір масиву, вказаного в попередньому параметрі. У функції **WNetCloseEnum** є тільки один параметр, в якому треба вказати ту ж змінну, що і була написана в останньому параметрі при відкритті переліку **WNetopenEnum**.

Якщо перелік пройшов успішно, то можна вивести одержану інформацію на екран. Для цього запускається цикл від 0 до кількості повернених значень **EntriesToGet**:

*for i := 1 to EntriesToGet do*

*Memo1.Lines.Add(string(ResourceBuffer[i].lpRemoteName));*

Усередині циклу додати в компонент **Memo1** рядок, що містить ім'я ресурсу. Ім'я одержаного відкритого ресурсу можна прочитати в змінної **ipRemoteName** структури **ResourceBuffer**. Єдине, що тут треба пам'ятати: ResourceBuffer [i] .ipRemoteName - це не рядок, тому цей параметр треба перетворити на рядок. Для цього використовується функція String():

*string(ResourceBuffer[i].ipRemoteName)*

Такий сканер расшаренних ресурсів сканує тільки одну вказану машину. Через це використання даної програми в мережі нереально.

Програму можна доповнити перебором. Єдиний недолік такого алгоритму сканування - дуже велика повільність в роботі. Це пов'язано не з самим алгоритмом, а із швидкістю виконання використовуваних функцій. Якщо сканованої адреси немає в мережі, то програма може втратити зайві 3-5 секунд в непотрібних пошуках. Це дуже багато, і це буде абсолютне безглуздою втратою часу. Щоб позбавитися цього недоліку, можна перед скануванням провести операцію ping вказаної IP-адреси. Пінг проходить достатньо швидко і зможе сказати нам, чи існує вказана адреса. Якщо він існує, то можна відкривати перебір ресурсів, лежачих у вільному доступі, інакше немає сенсу витрачати час на безглузді пошуки.

**Самостійна робота.**

Відредагувати програму так, щоб він перебирав декілька адрес підряд або брав їх із списку.

У звіті продемонструвати лістинг сканування для одного комп'ютера і лістинг сканування декількох адрес. А також відповідні скрін (результати сканування).

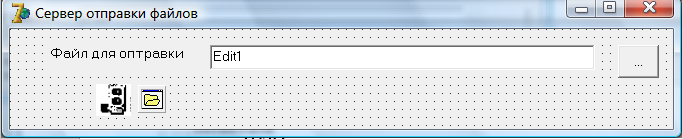
# Лабораторна робота №8

## Тема: Створення FTP-серверу.

Для того, щоб відправляти файли необхідно написати дві програми: клієнт і сервер. Сервер завантажуватиметься, відкриватиме порт і чекати з'єднання. Як тільки клієнт з'єднається і запитає у сервера файл, сервер вибере файл і відішле його.

Отже, для написання сервера треба запустити Delphi і для нового проекту перенести на форму наступні компоненти:

* поле введення **Edit**, в яке вводитиметься ім'я файлу;
* кнопку **Button**, за допомогою якої можна буде знаходити файл, що відправляється. Таким чином, не треба буде вводити повний шлях файлу уручну;
* компонент **ServerSocket**, з його допомогою ми відправлятиме дані;
* компонент **OpenDialog**, за допомогою якого ми відкриватимемо файл.



Мал. 7.1. Головна форма сервера

У компоненту **ServerSocket** потрібно встановити властивість **Port** рівним якому-небудь реальному значенню порту, який відкриватиметься для очікування підключення. Наприклад, номер 2024. Після чого встановити властивість **Active** рівним **true**, щоб сервер автоматично активізувався при старті.

У обробник натиснення кнопки потрібно вставити наступний код:

*OpenDialog1.Execute then*

*Edit1.Text:=OpenDialog1.FileName;*

Тут ми просто відображаємо на екрані вікно вибору файлу, і якщо користувач щось вибрав, то поміщаємо знайдене в рядок введення.

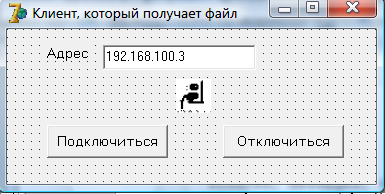
Тепер потрібно написати програму для роботи з мережею. Сервер повинен чекати приходу певної команди, і якщо вона прийшла, то відправляти файл. Для цього треба створити обробник події **onClientRead**, який викликається кожного разу, коли дані приходять з мережі. Спочатку перевіряється, що за команда прийшла. Якщо це буква **s**, то означає, клієнт просить прислати йому файл. Але перш ніж посилати дані, файл потрібно відкрити і завантажити. Для цього використовується об'єкт файлового потоку (змінна типу **TFileStream**). Спочатку ця змінна ініціалізувалася, а як параметри конструктора передається ім'я файлу **Edit1.Text** і режим, в якому буде підключений файл. Для цього досить використовувати режим читання (**fmOpenRead**). Після відкриття поточна позиція у файлі повинна бути встановлена на самий початок. Для перевірки встановлюємо позицію в початок. Тепер необхідно відправити клієнту розмір файлу. Розмір файлу можна дізнатНа початку даних, що відправляються, стоїть слово **Size:**, по якому клієнт дізнається, що ми вислали йому розмір файлу. Після цього йде сам розмір, перетворений в рядок. У самому кінці рядки додається нульовий символ #0, по якому клієнт зможе відокремити цю інформацію від даних самого файлу.

Тепер можна відправляти вибраний файл. Це краще зробити за допомогою методу **SendStream** компоненту **Socket**. Цей метод відправляє потік будь-якого формату (файловий потік, потік в пам'яті і т. д.).

Таким чином, сервер відправив файл, і можна приступати до написання клієнта, якого треба зробити окремою програмою.

Створюємо новий проект і поміщаємо на форму наступні компоненти:

* поле введення, куди ми вводитимемо IP-адресу сервера;
* кнопку **Підключитися**, при натисненні якої запрошуватиметься файл;
* кнопку **Відключитися**, за допомогою якої можна буде відключитися від сервера;
* компонент **TClientSocket**, за допомогою якого ми приєднуватимемося до сервера і запрашивать/получать файл.



Мал. 7.2. Головна форма клієнта

У компоненту **TClientSocket** потрібно встановити властивість **Port** рівним тому ж значенню, що і у сервера - 2024.

У розділі **private** об'єкту ми оголосимо дещо змінних:

*private*

*{Private declarations }*

*fs: TFileStream;*

*Reciving:Boolean;*

*DataSize: integer;*

У обробнику події **OnClick** кнопки **Підключитися** написати код, де в першому рядку встановлюється адреса комп'ютера, де розташований сервер, з яким треба з'єднатися. У другому рядку активізуємо клієнта і з'єднуємося.

У обробнику події **OnClick** кнопки **Відключитися** розташовуємо наступний код, просто закриваємо з'єднання.

Для створення обробника події **OnConnect** компоненту **ClientSocket**. Цей обробник викликатиметься тоді, коли клієнт встановить зв'язок з сервером. Як тільки ми з'єдналися з сервером, відразу ж відправляємо йому команду **s**, щоб сервер вислав файл.

Для створення обробника події **onRead** для компоненту **ClientSocket**, який викликатиметься кожного разу, коли клієнту приходять дані. Процедура достатньо складна і з нею доведеться розбиратися по частинах.

На самому початку зберігається прийнятий текст в змінній **s**. Наступна частина коду виконуватиметься, якщо змінна **Reciving** рівна **true**.

У наступній частині коду відбувається перевірка: якщо перші п'ять символів тексту, що прийшов, рівні слову **size:**, то означає, нас дійшов розмір файлу, і ми повинні почати його прийом. Спочатку вирізуємо розмір і збережемо його в текстовій змінній. Для цього необхідно скопіювати з тексту, що прийшов, всі символи від 6-го (після слова **size:**) і до символу **#0**:

*slt:=copy(s, 8, Pos(#0, s )-8 );*

Наступним рядком відбувається перетворення текстового представлення розміру в число і збереження його в змінній **DataSize**. Тепер з тексту, що прийшов, видаляємо всі символи до першого нульового символу **#0**, тобто видаляємо інформацію про розмір передаваного файлу.

Далі встановлюємо змінну **Reciving** рівної **true**. Ця змінна говоритиме про те, що почалася передача файлу. Оскільки файл не може прийти за один раз, і ми одержуватимемо його порціями приблизно по 8 Кбайт, то при подальших викликах цього обробника події треба знати, що що прийшло - це дані з файлу.

Дані, що залишилися, в змінній **s** - це вже перша порція файлу, яка була запитана користувачі. Щоб зберегти їх у файл, створюємо файловий потік:

*fs:=TFileStream. Create(‘output.dat’,fmCreate);*

Як ім'я указуємо, наприклад, таке ім'я файлу **output.dat**. Шлях не указується, означає, файл буде записаний в ту ж теку, де знаходиться програма. Як прапор при відкритті файлу указуємо **fmCreate**, що примушує створити новий файл. Якщо такий файл вже існує, то він буде перезаписаний.

Для того, щоб додати можливість передачі імені файлу сервер при передачі файлу повинен відправляти не тільки розмір, але і ім'я файлу.

Тепер на клієнті, після того, як ви виділили розмір файлу і видалили цей текст із змінної **s**, таким же чином можна виділити і ім'я файлу і видалити його з прийнятих даних, щоб зберігати у файл тільки його дані.

Після відкриття файлу зберігаємо прийняті дані за допомогою виклику методу **write**.

У наступній частині програми відбувається перевірка: якщо змінна **Reciving** рівна **true**, означає, відбувається прийом даних, і ми одержуємо чергову порцію файлу. Вона зберігається все тим же способом. Після цього відбувається порівняння, якщо розмір потоку рівний одержаному розміру файлу, то файл прийнятий повністю, і можна його закривати, вимикати змінну **Reciving** і виводити повідомлення про вдалий прийом.