МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп’ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Домерецький Сергій Михайлович

Система виявлення відкритих портів TCP/IP робочої станції / Workstation open TCP/IP ports Detection System

напрям підготовки: 6.050102 - Комп’ютерна інженерія
фахове спрямування - Комп’ютерні системи та мережі
Бакалаврська робота

Виконав студент групи КСМз-41/2
С.М. Домерецький

Науковий керівник:
Климчук О.І.

Тернопіль - 2018

ЗМІСТ

ВСТУП

Інтенсивний розвиток глобальних комп'ютерних мереж, поява нових технологій пошуку інформації привертають все більше уваги до мережі Internet з боку приватних осіб і різних організацій. Багато організацій приймають рішення про інтеграцію своїх локальних і корпоративних мереж в глобальну мережу. Використання глобальних мереж у комерційних цілях, а також при передачі інформації, яка містить відомості конфіденційного характеру, тягне за собою необхідність побудови ефективної системи захисту інформації. В даний час в Україні глобальні мережі застосовуються для передачі комерційної інформації різного рівня конфіденційності, наприклад для зв'язку з віддаленими офісами з головної штаб квартири організації або створення Web-сторінки організації з розміщеною на ній рекламою і діловими пропозиціями.

Навряд чи потрібно перераховувати всі переваги, які отримує сучасне підприємство, маючи доступ до глобальної мережі Internet. Але, як і багато інших нові технології, використання Internet має і негативні наслідки. Розвиток глобальних мереж призвело до багаторазового збільшення кількості користувачів і збільшення кількості атак на комп'ютери, підключені до мережі Internet. Щорічні втрати, зумовлені недостатнім рівнем захищеності комп'ютерів, оцінюються десятками мільйонів доларів. При підключенні до Internet локальної або корпоративної мережі необхідно подбати про забезпечення інформаційної безпеки цієї мережі. Глобальна мережа Internet створювалася як відкрита система, призначена для вільного обміну інформацією. У силу відкритості своєї ідеології Internet надає для зловмисників значно більші можливості в порівнянні з традиційними інформаційними системами. Тому питання про проблему захисту мереж і її компонентів стали досить важливими та актуальними в цей час, час прогресу комп'ютерних технологій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сканери портів

Досить часто системним адміністраторам і фахівцям з інформаційної безпеки необхідно з'ясувати наявність відкритих портів на досліджуваній операційній системі. Порти можуть бути відкриті програмним забезпеченням, вірусами, системними службами і так далі. Аналіз відкритих портів дозволяє ефективно боротися, наприклад, з вірусами, які можуть заразити систему і які не видно антивірусу.

Крім цього, інформація про відкриті портах дозволяє адміністратору відключити частину сервісів, які відкривають порти на прослуховування, і, тим самим, підвищити загальну захищеність системи.

Сканери портів - це спеціально розроблені програмні засоби для пошуку в мережі хостів з відкритими портами. Сканування може бути направлено на пошук відкритих портів в межах одного хоста і на пошук певного порту у всій мережі (принцип дії мережевих черв'яків).

Дані програми застосовують мережеві адміністратори для діагностики уразливості своєї мережі і попередження її злому, і хакери для проникнення в яких атакували мережі.

Сканування портів або мережі - це перший крок в процесі несанкціонованого доступу в мережу, визначення операційної системи, виявлення працюючих служб і сервісів.

Сканери портів в процесі роботи шукають найбільш часто використовувані і вразливі порти одного або декількох хостів. Результатом сканування будуть наступні категорії відповідей:

- порт відкритий - з'єднання з хостом може бути встановлено, даний стан порту може спричинити порушення стабільності функціонування сервісів і служб на хості;

- порт закритий - хост підтвердив, що несанкціоновані з'єднання будуть відкинуті, такий стан порту становить небезпеку для стабільної роботи операційної системи;

- порт заблокований - сканер не отримав жодної відповіді від порту, небезпеки для системи не несе.

Програмні засоби можуть сканувати мережі в наступних режимах:

- Перевірка хоста на мережеву активність (онлайн). Перший етап сканування - визначення роботи цільового IP-адреси шляхом відправки через утиліту ping Echo-повідомлення.

- Сканування SYN. При цьому типі сканування програма формує і відправляє SYN-пакети на цільової хост, аналізує відповіді і затримки відповідей.

- Сканування TCP. Найбільш простий і швидкий вид сканування портів. До переваг можна віднести відсутність спеціальних прав доступу, недоліком - велике навантаження на скановану системи і широкі можливості виявлення активності сканера.

- Сканування за допомогою відправки UDP-пакетів. Особливість даного типу сканування в тому, що деякі брандмауери невірно повідомляють про відкриття порту і в результаті може здаватися, що всі порти відкриті.

- АСК сканування показує наявність в системі брандмауера і визначає його налаштування.

Існують ще кілька способів сканування мережі, які мають свої плюси і мінуси, і призначені для отримання певного виду інформації про цільове хості або мережі.

Більшість мережевих екранів (фаєрволом) при правильному налаштуванні здатні захистити машину від сканування портів. Однак, хакерами розробляються все нові техніки злому систем і говорити про 100% захист неможливо.

Інтернет-провайдери забезпечують фільтрування вхідної та вихідної активності по скануванню портів, захищаючи своїх абонентів від несанкціонованого доступу до системи.

1.2 Аналіз програмних засобів для сканування портів

PortQry 2.0 сумісний з Windows 2003, XP і Windows 2000. Він відображає стан портів TCP і UDP на локальній або віддаленій машині. За функціональністю PortQry поступається більшості інших сканерів мережевих портів, таких як Nmap або безкоштовні SuperScan 4.0 і ScanLine компанії Foundstone. Але якщо запустити PortQry в локальному режимі на Windows 2003 або XP, то програма встановить відповідність між відкритими портами і мережевими додатками-слухачами аж до рівня DLL. Крім того, звіти PortQry більш насичені інформацією, ніж звіти Netstat.

При роботі в дистанційному режимі PortQry показує, чи закритий або відкритий даний порт. Для деяких портів, наприклад, LDAP (Lightweight Directory Access Protocol - спрощений протокол доступу до мережевих каталогів) і RPC, PortQry видає додаткову інформацію про прослуховуючу службу. Наприклад, при запиті LDAP до порту UDP 389 PortQry передає корисну інформацію про службу LDAP, в тому числі про її схему, конфігурацію і контекстне іменування кореневого домена; версії LDAP;хост-ім'я DNS.

PortQry - інструмент командного рядка. Однак окремо поширювана програма PortQueryUI, яка розглядається нижче, доповнена графічним інтерфейсом для PortQry і декількома функціями, що полегшують роботу з продуктом.

Щоб завантажити PortQryV2.exe, слід звернутися на сайт і вказати в рядку пошуку ім'я утиліти. Після завантаження аналізатора потрібно запустити саморозпаковуючийся архів, щоб відновити файли в будь-якому каталозі по

вибору користувача. Виконуваний файл portqry.exe дуже малий - всього 140 Кбайт. У командному рядку слід перейти до каталогу, в якому встановлена утиліта, і запустити portqry.exe, щоб відобразити всі параметри запуску. За допомогою PortQry можна досліджувати віддалені машини, але це повільний і примітивний процес в порівнянні з іншими аналізаторами портів. Наприклад, на відміну від Nmap, PortQry не дозволяє сканувати з використанням специфічних пакетних прапорів (SYN, FIN).

Щоб визначити, чи встановлений Web-сервер по TCP-порту 80 віддаленої машини з IP-адресою 192.168.0.8, слід виконати команду:

```
portqry -n 192.168.0.8 -e 80
```

Ключ -n задає ім'я або IP-адресу машини. Ключ -e вказує порт призначення або кінцеву точку і передує номеру порта. За замовчуванням PortQry виконує сканування з використанням протоколу TCP, але, як показано нижче, можна вказати і інший протокол. Якщо Web-сервер активний, то PortQry видає наступну інформацію:

```
TCP Port 80 (http service): LISTENING
```

За допомогою альтернативних джерел можна налаштувати поведінку PortQry. Ключі -r і -o забезпечують сканування діапазону послідовних портів або задають список окремих портів, відповідно:

```
portqry -r 10: 100
portqry -o 53,80,443
```

Порт-джерело задається ключем -sp, після якого вказується номер порту. Змінити обраний за замовчуванням протокол можна ключем -p, за яким слідують позначення UDP або BOTH. Якщо для сканування UDP-пакетів використовується повільний канал зв'язку, необхідно вказати затримку повільного каналу за допомогою ключа -sl і налаштувати PortQry на більш тривале очікування відповідей UDP від віддалених комп'ютерів.

Для сканування порту SNMP потрібно застосувати ключ -cn з подальшою рядком імені спільноти SNMP, обмеженою знаками оклику:

```
portqry -cn! community_string!
```

де community_string - ім'я спільноти SNMP. Якщо не вказати правильне ім'я спільноти, то замість відомостей по відкритому чи закритому порту буде отримано результат Filtered. За допомогою PortQry можна швидко перевірити, чи всі системи мають вірно задане ім'я спільноти, і визначити, чи є машини, налаштовані на прийом відомої рядки спільноти Public.

PortQry витягує корисну інформацію з певних портів. Наприклад, якщо запит направлений до порту відповідностей кінцевих точок RPC (TCP-порт 135), то PortQry повертає інформацію про всі служби, прослуховуючі цей порт.

Недоліки дистанційного сканування PortQry компенсуються унікальними локальними функціями. Для активізації локального режиму потрібно запустити PortQry з ключем -local. Якщо використовувався тільки ключ -local, то PortQry видає інформацію про використання всіх локальних портів і відповідностях порт-PID.

Замість сортування даних за відкритими портам, PortQry перераховує їх відповідно до PID. В результаті адміністратор може швидко з'ясувати, які програми мають у своєму розпорядженні відкритими мережевими з'єднаннями.

За допомогою PortQry можна відстежувати зміни стану конкретного локального порту або PID. Для спостереження за портом 80 потрібно виконати команду portqry -local -wport 80 і PortQry видасть список всіх додатків, прослуховуючих порт 80. Наприклад, якщо на системі працює Microsoft IIS, то PortQry повідомляє, що служба W3SVC знаходиться в стані Listening, і перераховує інші порти, використовувані службою W3SVC, зокрема порт 443 для HTTP Secure (HTTPS). Крім того, PortQry продовжує працювати в командній оболонці і видає інформацію про всі зміни активності порту.Таким чином, якщо потім відкрити браузер і звернутися до сервера, що працює з IIS, то PortQry повідомить про цю подію і нове з'єднання (рисунок 1.1).

Для спостереження за певним додатком слід запустити PortQry з ключем -wpid:

```
portqry -local -wpid PID
```

де PID - ідентифікатор процесу програми. Відстежуючи PID, легко контролювати мережеву активність нової програми. Припустимо, потрібно визначити, які порти використовує Windows Messenger. Відомо, що ім'я виконуваного файлу програми - msmsgs.exe, і його можна побачити в Task Manager. Слід запустити файл msmsgs.exe (або переконатися, що піктограма Windows Messenger знаходиться в системній панелі), потім запустити Task Manager. Клацнувши на вкладці Processes в Task Manager, можна переконатися, що в ній показаний стовпець PID.

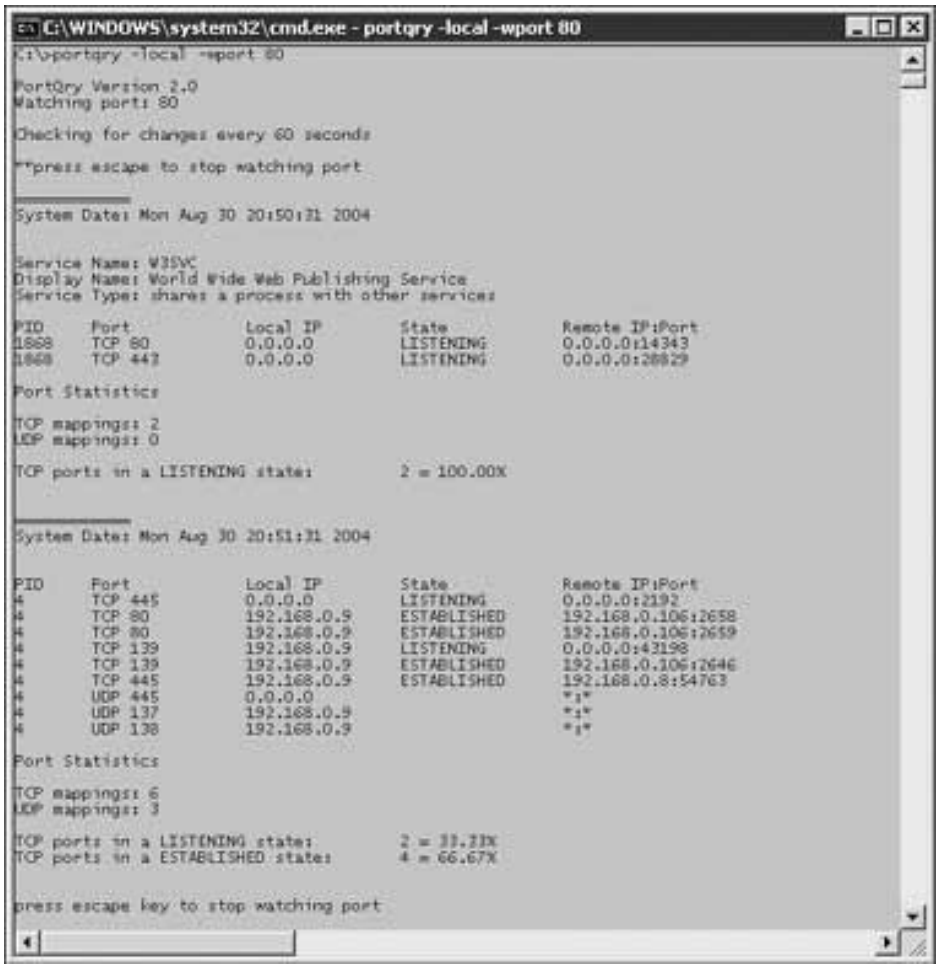

Рисунок 1.1 - Використання PortQry для спостереження за портом

Якщо стовпець відсутній, потрібно клацнути в меню View, Select Columns і встановити прапорець PID (Process Identifier). Необхідно відшукати процес msmsgs.exe і запам'ятати його PID, потім відкрити інше вікно командного рядка

і запустити PortQry з ключем -wpid і цим PID. Результат виконання програми можна побачити на рисунку 1.2.

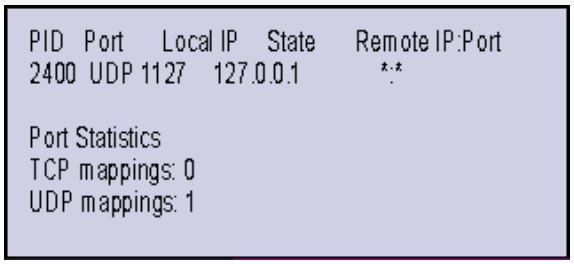

Рисунок 1.2 - Приклад виведення PortQry при спостереженні за процесом по його PID

Як показано на рисунку 1.2, процес з PID 2400 очікує UDP-порт 1127. Якщо залишатися в режимі спостереження, то PortQry виводить цю інформацію до тих пір, поки не буде виявлено зміна. PortQry перевіряє зміни кожну хвилину. Наприклад, якщо зареєструватися в Windows Messenger, то PortQry автоматично видає підсумкову зведення нових дій в командну оболонку (рисунок 1.*3*).

Тепер відомо, що Windows Messenger прослуховує порт UDP +1127 і використовує порт призначення TCP +1863 для зв'язку в процесі реєстрації в додатку. Слід зазначити, що Windows Messenger ініціював це з'єднання з віддаленим IP-адресом, 207.46.107.118, який відповідає IP-адресі, пов'язаній з вузлом msgr.hotmail.com, частиною мережі Windows Messenger.

Якщо послати діалогове повідомлення, виявляється цікавий факт. PortQry повідомляє про двоє виявлених сокетів (сокет - комбінація порту і віддаленої IP-адреси). Нові сокети - ще один порт з одержувачем TCP 1863 і новий порт з одержувачем TCP 80 (Рисунок 1.4).

TCP-порт 80 використовується протоколом HTTP і вказує, що Windows Messenger звертається до віддаленого Web-сервера при відправленні діалогового повідомлення. Через кілька хвилин відсутності активності сеанс TCP закінчується, порти закриваються, і PortQry повідомляє, що тільки два первинних порти залишаються відкритими.

На цьому досить простому прикладі наочно видно деякі переваги PortQry. За допомогою Task Manager і PortQry можна спостерігати за активністю певних програм і розпізнавати які порти використані.

Відомо, що Windows Messenger задіє TCP-порт 1863 але програма використовує і порт для HTTP. Крім того, складно спостерігати за конкретним додатком за допомогою традиційного аналізатора мережі, який реєструє трафік TCP / IP. Netstat видає аналогічну інформацію про відкриті портах, але не дозволяє відстежувати єдиний PID або повідомити тільки про зміни, пов'язані з новою активністю в мережі.

Microsoft пропонує графічний інтерфейс для PortQry (portqryui.exe). PortQryUI також можна завантажити з сайту. У PortQryUI входить версія portqry.exe і деякі заздалегідь певні набори, що складаються просто з груп портів для сканування. Ці набори - Domains and Trusts, IP Security (IPSec), Networking, SQL Service, Web Service, Exchange Server, NetMeeting і Miscellaneous. Вказані конкретні порти і протоколи, що становлять ці набори. Можна створити і власні групи портів для сканування, зберігши їх у спеціальному конфігураційному файлі XML.

При скануванні з використанням заздалегідь підготовленого набору PortQryUI пов'язує воєдино кілька запусків PortQry, а потім виводить результати на екран. Наприклад, якщо запросити Domains and Trusts, то інтерфейс послідовно ініціює сеанси сканування PortQry всіх портів, пов'язаних з Active Directory (AD), наприклад NetBIOS, DNS, Kerberos, LDAP, RPC.

Недолік PortQryUI полягає в тому, що програма не працює в локальному режимі, тому користувач не може спостерігати за PID і портами. Оскільки PortQryUI ініціює окремі перевірки PortQry для кожного порту, вихідні дані містять багато зайвої інформації, наприклад повторно вказується час початку сканування. Можна очистити результати, просто запустивши утиліту для командного рядка і вказавши конкретні порти. Наприклад, команда:

```
portqry -n 192.168.0.8 -o 135,
389,636,3268,3269,53,88,445,
```

```
137,138,139,42 -p both
```

видає стислий варіант результатів PortQryUI для служби Domains and Trusts. Команду необхідно вводити одним рядком без пробілів між номерами портів.

Таким чином, PortQry дозволяє оцінити мережеву активність систем Windows. Цей компактний, простий у використанні аналізатор особливо корисний в локальному режимі. Програма може стати відмінним доповненням до набору інструментів кожного адміністратора.

Інструментів для пошуку відкритих портів досить велика кількість. Розглянемо можливості невеликого сканера TCP портів DoScan. Відмінною особливістю цього сканера є його компактність (розмір дистрибутива 8 мегабайт) і портативність (сканер не вимагає установки і може переноситися між різними комп'ютерами без втрати накопиченої статистики та налаштувань). DoScan працює на всіх версіях операційних систем Windows, починаючи з 2000. В згорнутому стані (це основний режим його постійної роботи) займає всього 2,5 мегабайта оперативної пам'яті. Під час сканування практично не навантажує операційну систему.

TCP сканер DoScan поширюється безкоштовно. Завантажити TCP сканер можна з сайту WindowsFAQ.com.

Для запуску необхідно розпакувати скачаний архів в будь-яку папку і запустити виконуваний файл DoScan.exe. Головне вікно програми показано на рисунку 1.3.

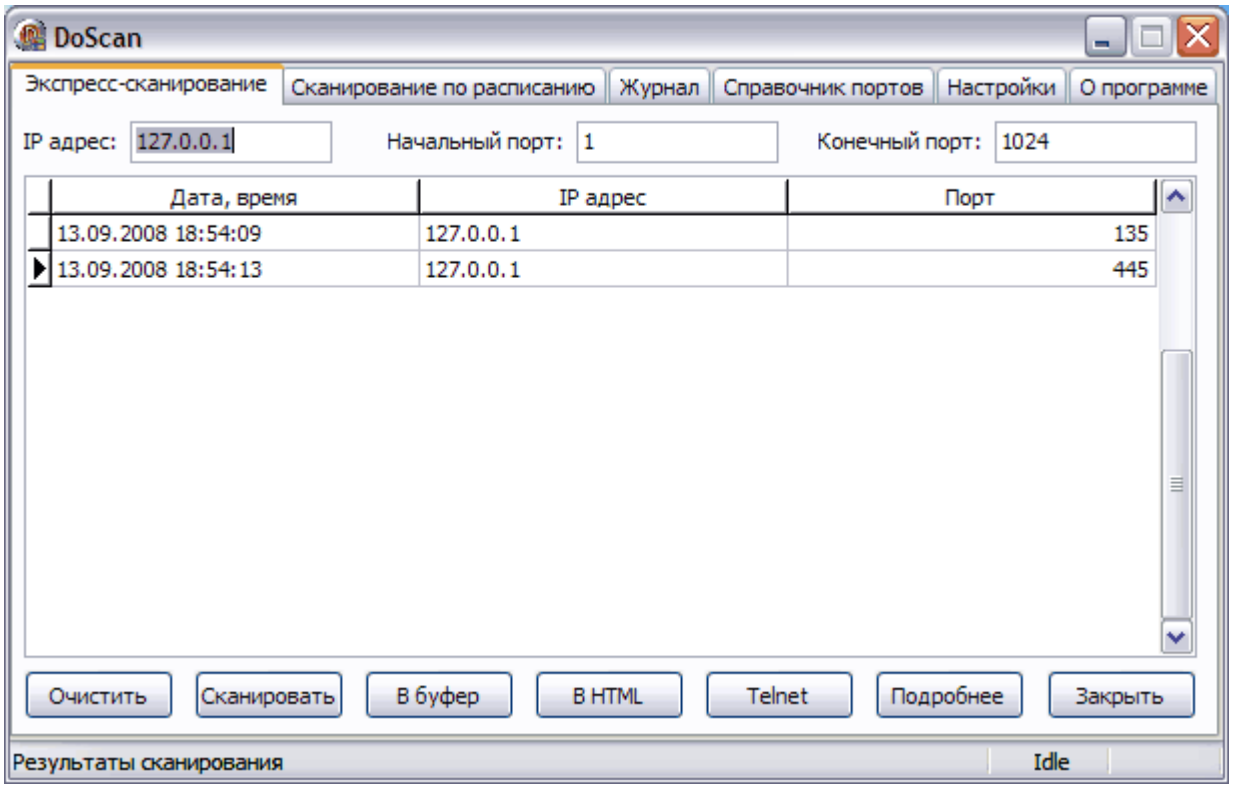

Рисунок 1.3 - Головне вікно сканера TCP портів DoScan

Сканер може виконувати сканування певного хоста на вимогу або за розкладом. На вкладці «Експрес-сканування», показаної на рисунку 1.3, налаштовуються параметри сканування на вимогу.

В ході роботи, інформація про кожен знайдений відкритий порт на досліджуваній машині, буде додана в таблицю. При необхідності ця таблиця може бути експортована в файл формату html. Подвійне клацання по запису в таблиці відкриває вікно з докладною інформацією про стандартні службах, які можуть використовувати знайдений відкритий порт (рисунок 1.4).

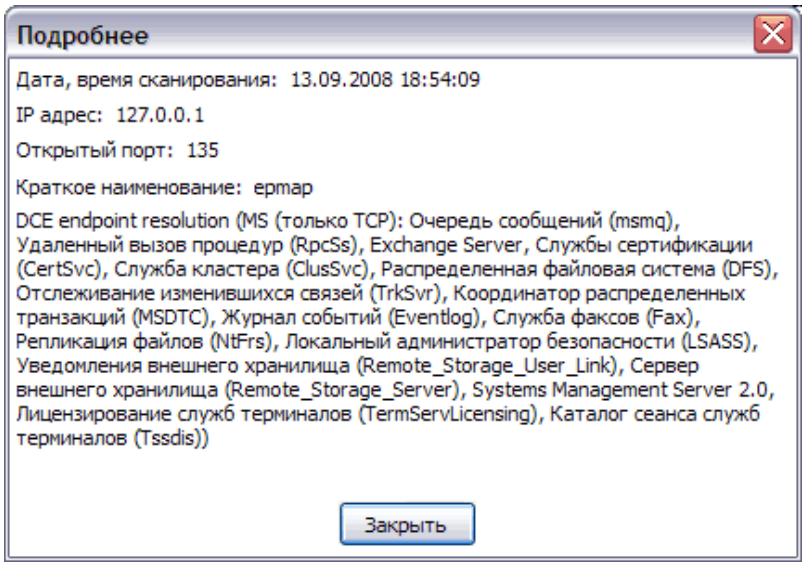

Рисунок 1.4 - розширена інформація

Якщо є необхідність регулярно отримувати інформацію про відкриті портах на будь-якій кількості комп'ютерів, то їх сканування можна запланувати на вкладці «Сканування за розкладом», як це показано на рисунку 1.5.

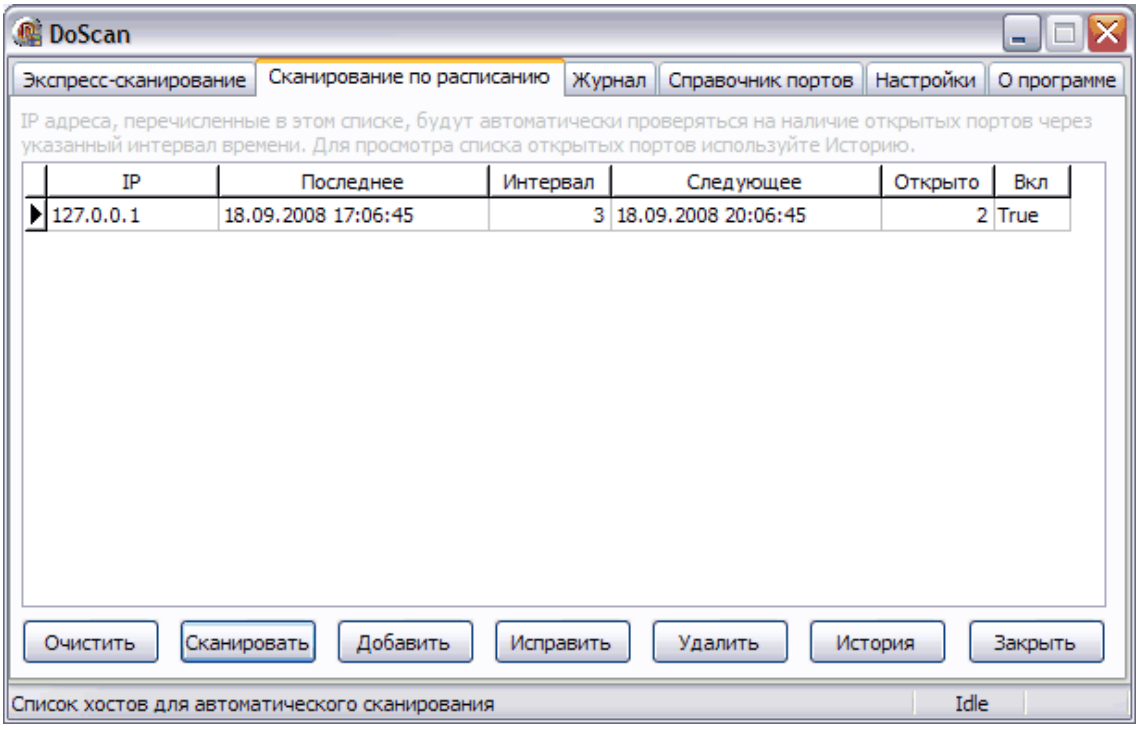

Рисунок 1.5 - Налаштування сканування за розкладом

Кожен з перерахованих в цьому списку хостів буде просканований через вказаний в його настройках інтервал часу (рисунок 1.6.).

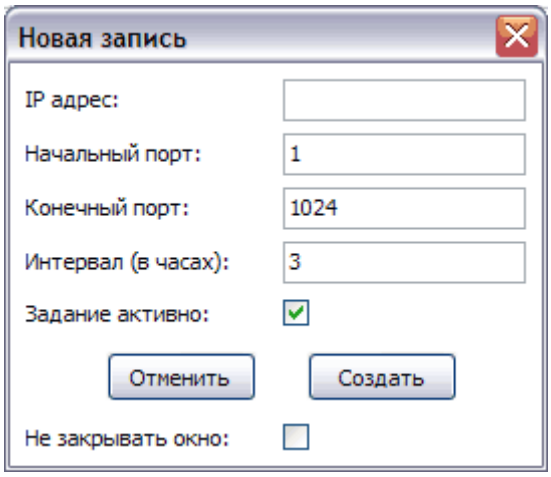

Рисунок 1.6 - Додавання нового хоста

У вікні додавання нового хоста (або зміни властивостей раніше створеного) вказується сканована IP адреса (буде просканований незалежно від доступності), початковий і кінцевий номери портів, інтервал між скануваннями в годинах. Завдання може бути створено неактивним і в цьому випадку воно не буде запущено. Якщо необхідно додати в розклад відразу кілька хостів, то можна відзначити перемикач «Тримати вікно» і після натискання кнопки «Зберегти» буде запропоновано створити наступний хост з настройками, аналогічними попередньому.

Після того, як список хостів для автоматичного сканування сформований, сканер буде його періодично переглядати і коли настане час сканування чергового хоста, завдання буде запущено. Щоб не перевантажувати мережу і не займати багато системних ресурсів, сканер одночасно може обстежити тільки один хост. Результати перевірки заносяться в журнал «Історія» і можуть бути переглянуті для будь-якого хоста. При видаленні хоста з розкладу автоматичного сканування, вся його історія сканувань так само видаляється.

Якщо якийсь хост на час необхідно виключити з автоматичного сканування, або потрібно зберегти його історію, то завдання для його автоматичної перевірки можна відключити.

Сканування будь-якого хоста, внесеного в список автоматичної перевірки, можна запустити вручну. Або, його налаштування можна

автоматично скопіювати на вкладку «Експрес-сканування» і запустити перевірку звідти (рисунок 1.7).

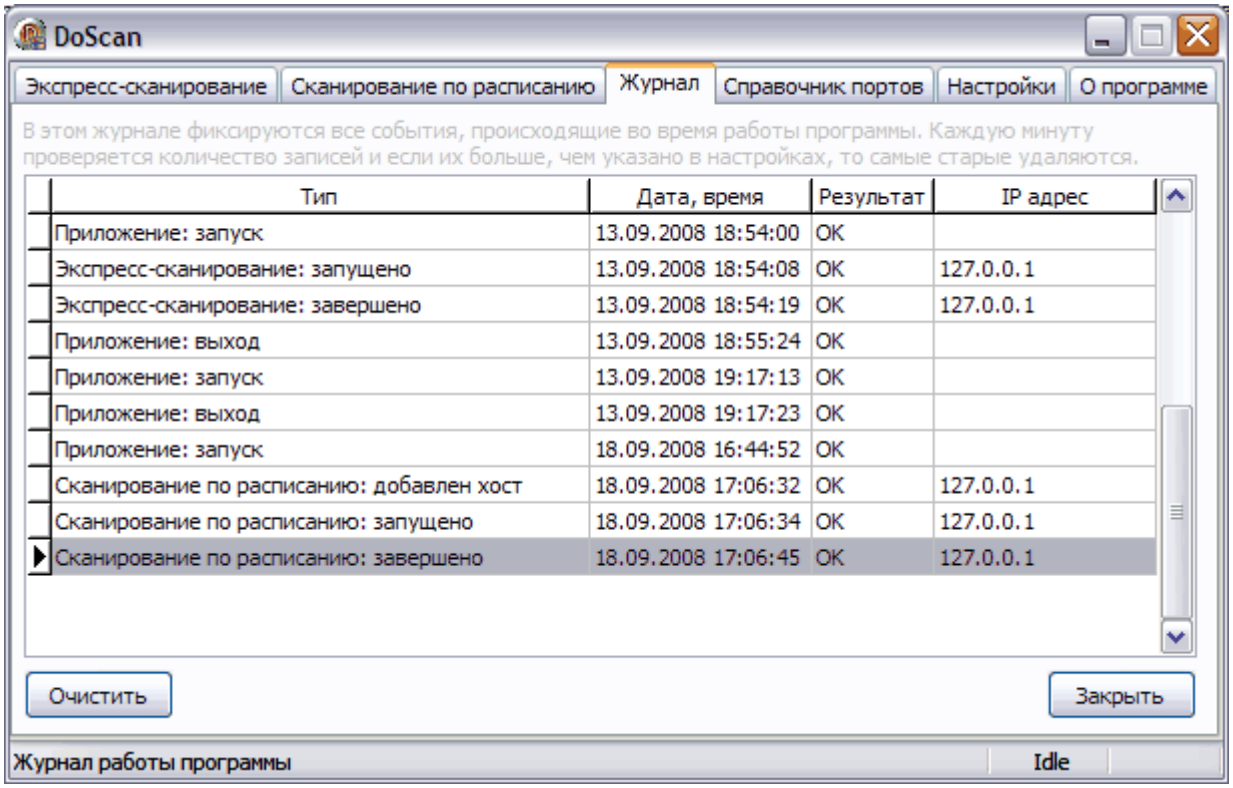

Рисунок 1.7 - Журнал роботи програми

Всі події, що відбуваються під час роботи DoScan, заносяться в журнал, показаний на рисунку 1.7. В тому числі, час запуску і результат як автоматичних, так і ручних запусків сканування.

Також ыснуэ можливысть перегляду довыдника портів, як показано на рисунку 1.8. На вкладці «Довідник портів» доступний список, в якому дані короткі найменування служб, зазвичай використовують певний порт, і повний його опис. Знайшовши відкритий порт на системі, завжди можна швидко отримати інформацію про службу, яка може використовувати цей порт.

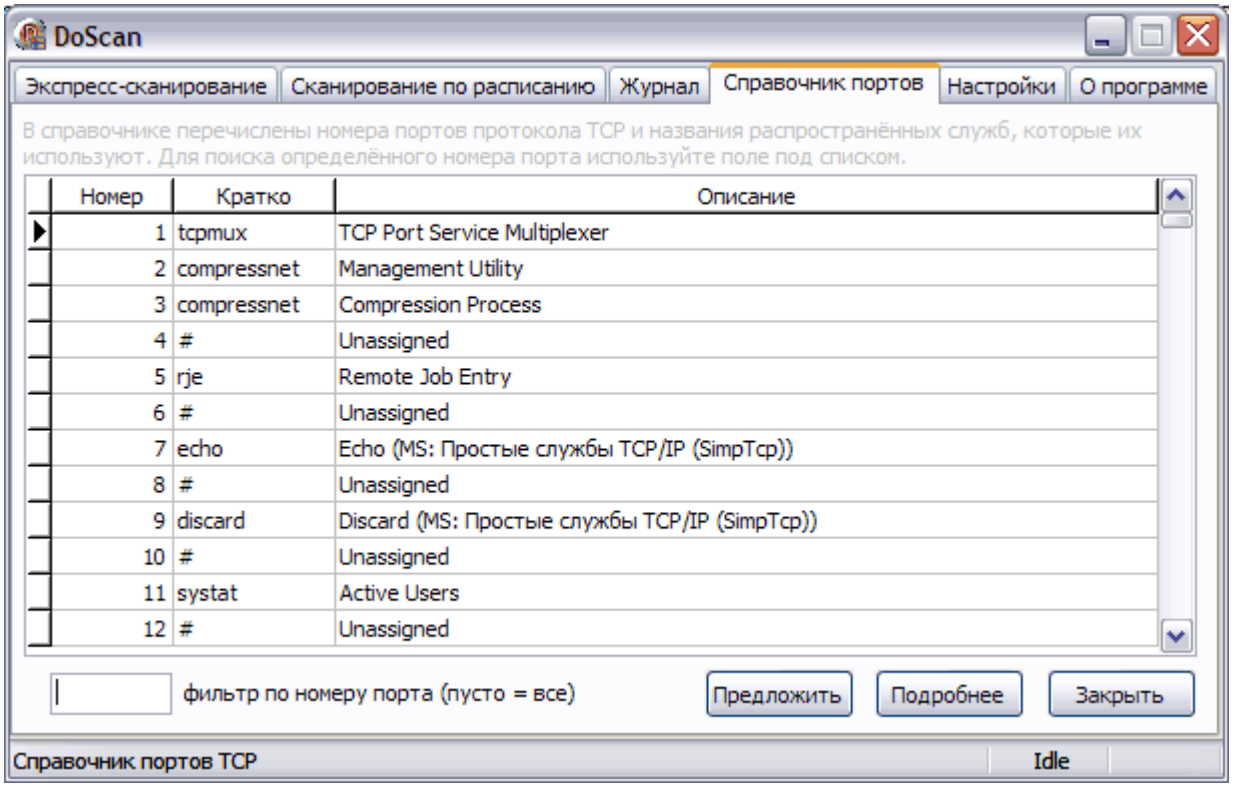

Рисунок 1.8 - Довідник портів TCP

Для полегшення пошуку певного номера порту, можна скористатися фільтром. За допомогою кнопки «Запропонувати» є можливість автоматично сформувати заготовку електронного листа розробнику DoScan, в якому описати порт, відсутній в списку, або запропонувати доповнення до опису. Ця інформація буде включена розробником в чергове оновлення DoScan.

Налаштування програми показані на рисунку 1.9. Якщо сканер згорнутий в трей, то інформація про запуск сканувань, знайдених відкритих портах і результати сканування можуть виводитися в підказках. Сканер може автоматично завантажуватися при внесені логіна користувача і згортатися в трей. Журнал результатів експрес-сканування може автоматично очищатися при запуску нового сканування, IP адреса сканируемого хоста може зберігатися між перезапуск сканера.

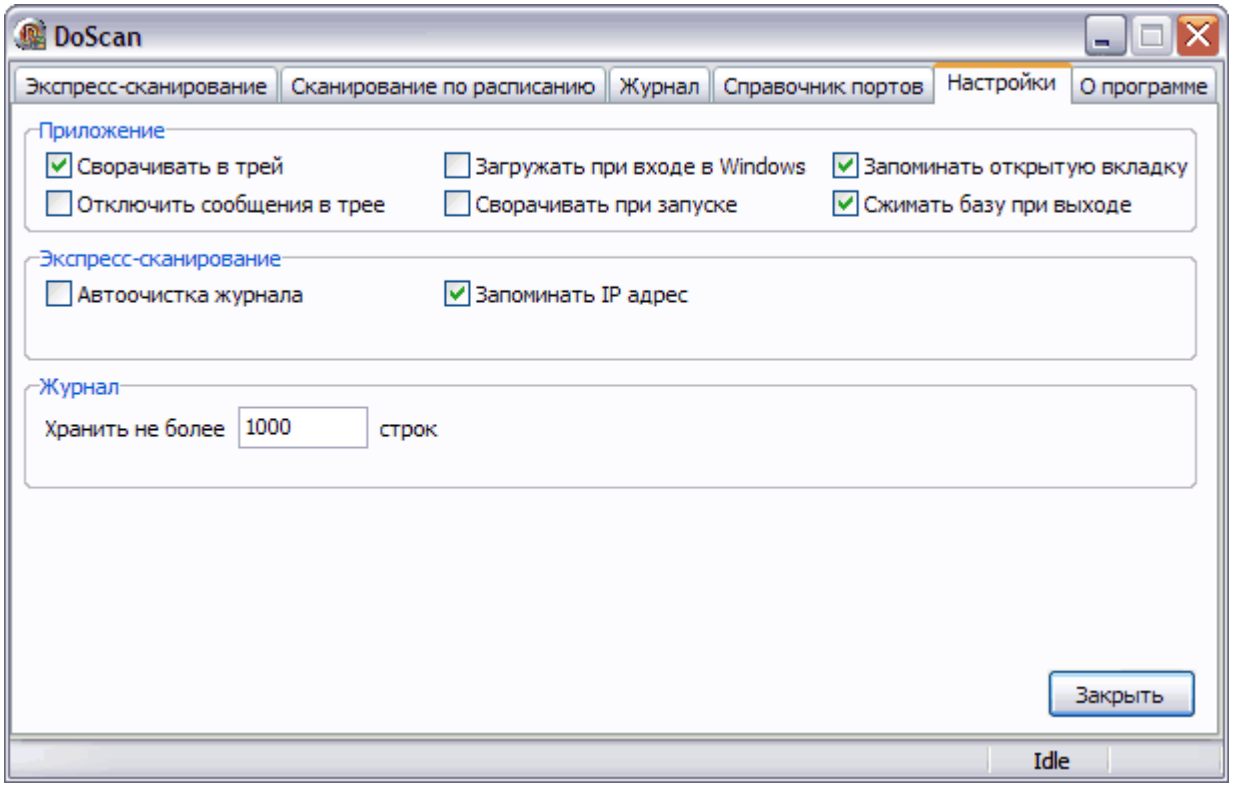

Рисунок 1.9 - Налаштування DoScan

На вкладці «Про програму» створена форма для відправлення листа розробнику. З її допомогою можна задати питання або внести пропозицію про функціонал майбутніх версій. На цій же вкладці є посилання для перевірки наявності оновлення та посилання на форум підтримки продукту.

1.3 Аналіз технічного завдання та постановка цілей проектування

Для реалізації завдання необхідно написати програмний продукт який буде забезпечувати виявлення відкритих портів.

Розробка програмного засобу зводиться до наступних етапів:

1. Розробка графічного інтерфейсу та класів для роботи з ним.
2. Розробка основних методів роботи з протоколами та відкритими портами.
3. Розробка основної функції для перебору по діапазону хостів .
4. Аналіз та виведення результатів скнування.

5. Розробка планувальника сканування.

Для реалізації цілей дипломного проектування та технічного завдання необхідно провести аналіз предметної області, дослідити існуючі методи виявлекння відкритих портів, різновиди портів, виділити їхні функціональні обмеження, недоліки та переваги. Для проектування програмного продукту потрібно дослідити алгоритмічне та програмне забезпечення, що будуть використанні при розробці. Необхідно провести проектування інтерфейсу, розробку основних функціональних можливостей додатку. Провести тестування та верифікацію, що дозволять підтвердити повноцінність і цілісність програмного продукту.

2 ФУНКЦІЇ РОБОТИ З СТЕКОМ ПРОТОКОЛІВ

2.1 Дослідження основ TCP / IP

TCP / IP - це абревіатура терміну Transmission Control Protocol / Internet Protocol (протокол управління передачею / протокол Internet). У термінології обчислювальних мереж протокол - це заздалегідь узгоджений стандарт, який дозволяє двом комп'ютерам обмінюватися даними. Фактично TCP / IP не один протокол, а декілька. Саме тому його називають набором, або комплектом протоколів, серед яких TCP і IP - два основних.

Програмне забезпечення для TCP / IP, на комп'ютері, являє собою специфічну для даної платформи реалізацію TCP, IP і інших членів сімейства TCP / IP. Зазвичай в ньому також є такі високорівневі прикладні програми, як FTP (File Transfer Protocol, протокол передачі файлів), які дають можливість через командний рядок управляти обміном файлами по Мережі.

TCP / IP - зародився в результаті досліджень, профінансованих Управлінням перспективних науково-дослідних розробок (Advanced Research Project Agency, ARPA) уряду США в 1970-х роках. Цей протокол був розроблений з тим, щоб обчислювальні мережі дослідницьких центрів у всьому світі могли бути об'єднані в формі віртуальної "мережі мереж" (internetwork). Первісна Internet мережа була створена в результаті перетворення існуючого конгломерату обчислювальних мереж, що звався ARPAnet, за допомогою TCP / IP.

Причина, по якій TCP / IP настільки важливий сьогодні, полягає в тому, що він дозволяє самостійним мережам підключатися до Internet або об'єднуватися для створення приватних мереж. Обчислювальні мережі, складові, фізично підключаються через пристрої, які називаються маршрутизаторами або IP-маршрутизаторами. Маршрутизатор - це комп'ютер, який передає пакети даних з однієї мережі в іншу. В інтрамережі, що працює на основі TCP / IP, інформація передається у вигляді дискретних блоків, званих IP-

пакетами (IP packets) або IP-дейтаграммами (IP datagrams). Завдяки програмному забезпеченню TCP / IP всі комп'ютери, підключені до обчислювальної мережі, стають "локальними". По суті приховуються маршрутизатори та базові архітектури мереж, це все виглядає як одна велика мережа. Точно так, як підключення до мережі Ethernet розпізнаються по 48-розрядним ідентифікаторам Ethernet, підключення до локальної мережі ідентифікуються 32-розрядними IP-адресами, які виражаються у формі десяткових чисел, розділеними крапками (наприклад, 128.10.2.3). Взявши IP-адреса віддаленого комп'ютера, комп'ютер в інтрамережі або в Internet може відправити дані на нього, так ніби вони складають частину однієї і тієї ж фізичної мережі.

TCP / IP дає вирішення проблеми даними між двома комп'ютерами, підключеними до однієї і тієї ж локальної мережі, але належать різним фізичним мереж. Рішення складається з декількох частин, причому кожен член сімейства протоколів TCP / IP вносить свою лепту в спільну справу. IP - самий фундаментальний протокол з комплекту TCP / IP - передає IP-дейтаграми по інтрамережі і виконує важливу функцію, так звану маршрутизацією, по суті справи це вибір маршруту, за яким дейтаграмма буде слідувати з пункту A в пункт B, і використання маршрутизаторів для "стрибків "між мережами.

TCP - це протокол більш високого рівня, який дозволяє прикладним програмам, запущеним на різних головних комп'ютерах мережі, обмінюватися потоками даних. TCP поділяє потоки даних на ланцюжки, які називаються TCP-сегментами, і передає їх за допомогою IP. У більшості випадків кожен TCP-сегмент пересилається в одній IP-дейтаграми. Однак при необхідності TCP буде розщеплювати сегменти на кілька IP-дейтаграм, що вміщаються в фізичні кадри даних, які використовують для передачі інформації між комп'ютерами в мережі. Оскільки IP не гарантує, що дейтаграми будуть отримані в тій же самій послідовності, в якій вони були послані, TCP здійснює повторну "збірку" TCP-сегментів на іншому кінці маршруту, щоб утворити безперервний потік даних.

FTP і telnet - це два приклади популярних прикладних програм TCP / IP, які спираються на використання TCP.

Інший важливий член комплекту TCP / IP - User Datagram Protocol (UDP, протокол призначених для користувача дейтаграм), який схожий на TCP, але більш примітивний. TCP - "надійний" протокол, тому що він забезпечує перевірку на наявність помилок і обмін підтверджуючими повідомленнями щоб дані досягали свого місця призначення завідомо без спотворень. UDP - "ненадійний" протокол, бо не гарантує, що дейтаграми будуть приходити в тому порядку, в якому були послані, і навіть того, що вони прийдуть взагалі. Якщо надійність - бажана умова, для його реалізації буде потрібно програмне забезпечення. Але UDP як і раніше займає своє місце в світі TCP / IP, і використовується в багатьох програмах. Прикладна програма SNMP (Simple Network Management Protocol, простий протокол управління мережами), що реалізується в багатьох втіленнях TCP / IP, - це один із прикладів програм UDP.

Інші TCP / IP протоколи грають менш помітні, але в рівній мірі важливі ролі в роботі мереж TCP / IP. Наприклад, протокол визначення адрес (Address Resolution Protocol, ARP) перетворює IP-адреси в фізичні мережеві адреси, такі, як ідентифікатори Ethernet. Протокол зворотного перетворення адрес (Reverse Address Resolution Protocol, RARP) - виконує зворотну дію, перетворюючи фізичні локальну мережу в IP-адреси. Протокол управління повідомленнями Internet (Internet Control Message Protocol, ICMP) являє собою протокол супроводу, який використовує IP для обміну керуючою інформацією і контролю над помилками, які належать до передачі пакетів IP. Наприклад, якщо маршрутизатор не може передати IP-дейтаграмму, він використовує ICMP, з тим щоб інформувати відправника, що виникла проблема.

Короткий опис протоколів сімейства TCP / IP з розшифровкою абревіатур:

- ARP (Address Resolution Protocol, протокол визначення адрес): конвертує 32-розрядні IP-адреси в фізичні адреси обчислювальної мережі, наприклад, в 48-розрядні адреси Ethernet.

- FTP (File Transfer Protocol, протокол передачі файлів): дозволяє передавати файли з одного комп'ютера на інший з використанням TCP-з'єднань. Існує аналог побудований на основі FTP протоколу, який є також поширеним протоколом передачі файлів - Trivial File Transfer Protocol (TFTP) - для пересилки файлів застосовується UDP, а не TCP.

- ICMP (Internet Control Message Protocol, протокол керуючих повідомлень Internet): дозволяє IP-маршрутизаторам посилати повідомлення про помилки і керуючу інформацію іншим IP-маршрутизатам і головним комп'ютерам мережі. ICMP-повідомлення "маршрутизуються" у вигляді полів даних IP-дейтаграм і обов'язково повинні реалізовуватися у всіх варіантах IP.

- IGMP (Internet Group Management Protocol, протокол управління групами Internet): дозволяє IP-дейтаграмам поширюватися в циркулярному режимі (multicast) серед комп'ютерів, які належать до відповідних груп.

- IP (Internet Protocol, протокол Internet): низькорівневий протокол, який направляє пакети даних по окремих мережах, пов'язаних разом за допомогою маршрутизаторів для формування Internet або інтрамережі. Дані "маршрутизуються" у формі пакетів, званих IP-дейтаграммами.

- RARP (Reverse Address Resolution Protocol, протокол зворотного перетворення адрес): перетворить фізичні мережні адреси в IP-адреси.

- SMTP (Simple Mail Transfer Protocol, простий протокол обміну електронною поштою): визначає формат повідомлень, які SMTP-клієнт, що працює на одному комп'ютері, може використовувати для пересилки електронної пошти на SMTP-сервер, запущений на іншому комп'ютері.

- TCP (Transmission Control Protocol, протокол управління передачею): протокол орієнтований на роботу з підключеннями і передає дані у вигляді потоків байтів. Дані пересилаються пакетами - TCP-сегментами, - які складаються з заголовків TCP і даних.TCP - "надійний" протокол, тому що в

ньому використовуються контрольні суми для перевірки цілісності даних і відправка підтверджень, щоб гарантувати, що передані дані прийняті без спотворень.

- UDP (User Datagram Protocol, протокол призначених для користувача дейтаграм): протокол, що не залежить від підключень, який передає дані пакетами, званими UDP-дейтаграммами. UDP - "ненадійний" протокол, оскільки відправник не одержує інформацію, яка показує, чи була в дійсності прийнята дейтаграма.

Проектувальники обчислювальних мереж часто використовують семирівневу модель ISO / OSI (International Standards Organization / Open Systems Interconnect, Міжнародна організація по стандартизації / Взаємодія відкритих систем), яка описує архітектуру мереж. Кожен рівень в цій моделі відповідає одному рівню функціональних можливостей мережі. У самій основі розташовується фізичний рівень, який представляє фізичне середовище, по якій "маршрутизуються" дані, - іншими словами, кабельну систему обчислювальної мережі. Над ним є канальний рівень, або рівень ланки даних, функціонування якого забезпечується мережевими інтерфейсними платами. На самому верху розміщується рівень прикладних програм, де працюють програми, що використовують службові функції мереж.

На рисунку 2.1 показано, як TCP / IP узгоджується з моделлю ISO / OSI. Цей рисунок також ілюструє рівневу будову TCP / IP і показує взаємозв'язок між основними протоколами. При перенесенні блоку даних з мережевої прикладної програми в плату мережевого адаптера він послідовно проходить через ряд модулів TCP / IP.

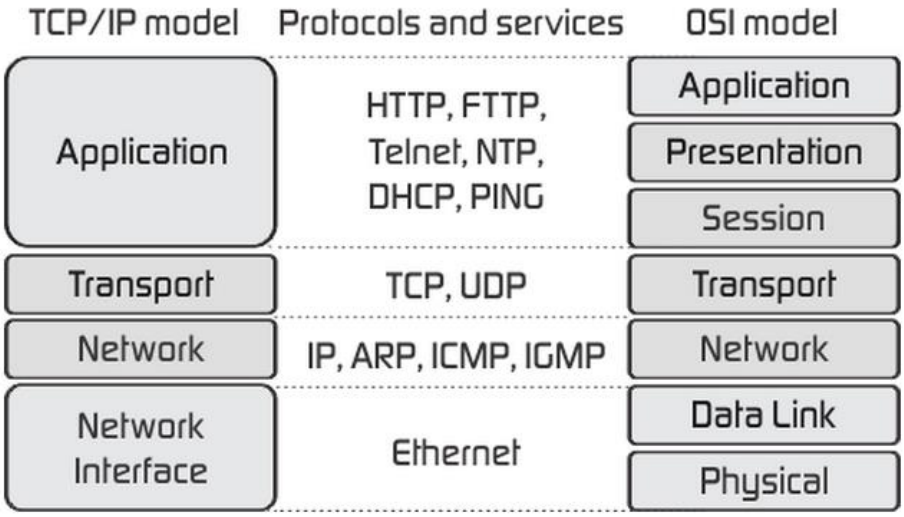

Рисунок 2.1 - Рівні мереж і протоколи TCP / IP ISO / OSI TCP / IP

При цьому на кожному кроці він має бути доповнений інформацією, необхідною для еквівалентного модуля TCP / IP на іншому кінці ланцюжка. До того моменту, коли дані потрапляють в мережеву плату, вони представляють собою стандартний кадр Ethernet, якщо припустити, що мережа заснована саме на цьому інтерфейсі. Програмне забезпечення TCP / IP на приймальному кінці відтворює вихідні дані для приймаючої програми шляхом захоплення кадру Ethernet і проходження його в зворотному порядку по набору модулів TCP / IP.

2.2 Програмування сокетів в Delphi

У Delphi існують вбудовані класи для роботи з мережею - це компоненти Delphi 6 на закладці Internet (TServerSocket і TClientCocket) і компоненти FastNet, або компоненти Indy в Delphi 7. Дослідимо програмування мережевих додатків на низькому рівні - з використанням WinSock API.

Існує великий набір вбудованих компонентів і велика кількість безкоштовно розповсюджуються класів, проте:

1. У разі, якщо для додатка критичний розмір (це найбільше відноситься до серверних додатків, які найчастіше не мають ніякої візуальної частини), то використання будь-яких VCL-компонентів вкрай небажано.

2. Програмування на більш низькому рівні традиційно вимагає великих витрат, але є кращою в для того щоб отримати повний контроль за роботою програми.

Для підтримки мережних додатків існує технологія, названа "сокети". Сокет - це модель одного кінця з'єднання, з усіма притаманними йому властивостями і методами. По суті, це прикладний програмний інтерфейс, що входить до складу багатьох операційних систем (ОС) і покликаний для підтримки мережевих можливостей ОС. У стандарті структури протоколів семирівневої моделі OCI лежать на так званому транспортному рівні, нижче знаходиться мережевий протокол IP, а вище - протоколи сеансового рівня, такі як FTP, POP3, SMTP і т.д.

У Windows підтримка сокетів включена починаючи з версії 3.11 і названа WinSock. Для створення програмного забезпечення з мережевою підтримкою існує спеціальний WinSock API.

Сокети можуть базуватися на TCP / IP, IPX / SPX але надалі будемо розглядати тільки з'єднання TCP / IP.

Всі мережеві додатки побудовані на технології клієнт-сервер, це означає, що в мережі існує принаймні один додаток, що виконує роль сервера, типова задача якого - це очікування запиту на підключення від програм-клієнтів, яких може бути теоретично скільки завгодно, і виконання всіляких процедур у відповідь на запити клієнтів. Для клієнт-серверної технології абсолютно неважливо, де розташовані клієнт і сервер - на одній машині або на різних. Звичайно, для успішного з'єднання клієнта з сервером клієнтові необхідно мати мінімальний набір даних про розташування сервера - для мереж TCP / IP це IP-адреса комп'ютера, де розташований сервер, і адреса порту, на якому сервер очікує запити від клієнтів.

Кожен з комп'ютерів в мережі TCP / IP має свою унікальну IP-адресу, яка використовується для обміну даними з іншими комп'ютерами. Кожен пакет посилається від одного комп'ютера іншому має адресу відправника і одержувача, що дозволяє його однозначно ідентифікувати. Однак в разі, якщо на комп'ютері працює безліч додатків, одночасно використовують мережу, то такого набору атрибутів явно недостатньо.

Для вирішення неоднозначності крім адреси кожне з'єднання на кожному кінці має ідентифікатор під назвою "порт", цей ідентифікатор представляє число від 0 до 65535. Таким чином, пара адреса + порт являє собою сокет-канал, по якому два комп'ютери обмінюються даними один з одним. Тільки один додаток на одному комп'ютері в один і той же час може використовувати конкретний порт, однак для серверних частин можливе створення декількох сокетов на одному порту для роботи з декількома клієнтами.

Значення порту не обов'язково має збігатися на сервері і клієнті - клієнту для з'єднання важливо тільки знати порт сервера, порт клієнта може вибиратися клієнтом довільно і стає відомий серверу в момент запиту клієнта на з'єднання. Коли з'єднання буде встановлено, ОС створить для серверного додатка відповідний сокет, з яким і буде працювати додаток, так що порт клієнта для сервера абсолютно не важливий.

Механізм роботи сокетів такий: на серверній стороні запускається серверний сокет, який після запуску відразу переходить в режим прослуховування (тобто очікування з'єднання клієнтів). На стороні клієнта створюється сокет, для якого вказується IP-адреса і порт сервера і дається команда на з'єднання. Коли сервер отримує запит на з'єднання, ОС створює новий екземпляр сокета, за допомогою якого сервер може обмінюватися даними з клієнтом. При цьому сокет, який створений для прослуховування, продовжує перебувати в режимі прийому з'єднань, таким чином програміст може створити сервер, який працює з декількома підключеннями від клієнтів.

Робота з сокетами, по суті, це операції введення-виведення, які бувають синхронні і асинхронні. У термінології сокетів робота в асинхронному режимі

називається блокуючими сокетами, а в синхронному - неблокуючі сокети. Спроба з'єднання або прийому даних в блокуючому режимі (відправка завжди синхронна, так як фактично є постановкою в чергу) означає, що поки програма не з'єднається або не прийме дані, передачі управління на наступний оператор не відбудеться.

2.3 Дослідження WinSock API

Розглянемо мінімальний набір функцій з WinSock API, необхідних для написання елементарного клієнта і сервера. Самі функції знаходяться в файлі winsock.dll. Файл winsock.pas містить необхідні оголошення імпортованих функцій WinSock API і базові структури даних. Цей файл імпортує не всі необхідні функції, і пізніше потрібно написати свій файл імпорту.

function WSAStartup (wVersionRequired: word; var WSData: TWSAData): Integer; stdcall; - Функція повідомляє ОС, що в будь-якому процесі додатки можуть бути використані функції WinSock. Функція повинна бути викликана один раз при запуску програми перед використанням будь-якої функції WinSock.

function WSACleanup: Integer; stdcall; - Функція повідомляє ОС, що додаток більш не використовує WinSock. Повинна бути викликана перед завершенням програми.

function socket (af, Struct, protocol: Integer): TSocket; stdcall; - Функція створює сокет. Порт і адреса задається в функції bind (сервер) або connect (клієнт). Вхідний параметр af - специфікація сімейства сокетов (AF_INET, AF_IPX і ін.), Struct - специфікація типу нового сокета (приймає значення SOCK_STREAM або SOCK_DGRAM), protocol - специфічний протокол, який буде використовуватися сокетом. Якщо функція виконана без помилок, вона

повертає дескриптор на новий сокет, якщо помилки є, повертається INVALID_SOCKET.

function connect (s: TSocket; var name: TSockAddr; namelen: Integer): Integer; stdcall; - Функція з'єднання для клієнта. Структура адреси містить порт (необхідно привести функцією htons) і адреса (для клієнта необхідно привести з імені або специфікації ip4 - xxx.xxx.xxx.xxx).

function bind (s: TSocket; var addr: TSockAddr; namelen: Integer): Integer; stdcall; - Функція асоціює адресу з сокетом. Структура адреси містить порт (необхідно привести функцією htons) і адреса (для сервера зазвичай вказується INADDR_ANY - будь-який).

function send (s: TSocket; var Buf; len, flags: Integer): Integer; stdcall; - Функція відправки даних. Поміщає в чергу сокета s частину даних з buf, довжиною len. Останній параметр відповідає за вид передачі повідомлення. Може бути проігнорований (0).

function recv (s: TSocket; var Buf; len, flags: Integer): Integer; stdcall; - Функція отримання даних.

Було зпроектовано програмний засіб з кодом елементарного сервера і клієнта. Сервер буде працювати в асинхронному (блокуючому режимі). Єдина функціональність сервера - це отримання рядків даних від клієнта і виведення їх на екран. Зв'язок клієнта з сервером розривається після отримання рядка, що складається з єдиного символу 'q'.

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Розробка інтерфейсу програмного засобу

При проектуванні програмного засобу було використано наступні модулі: Windows, Messages, SysUtils, Classes, Controls, Forms, StdCtrls, ExtCtrls, WinSock, AppEvnts, Buttons, Gauges, Spin.

При розробці програмного засобу було створено інтерфейс користувача, який представлений на рисунку 3.1.

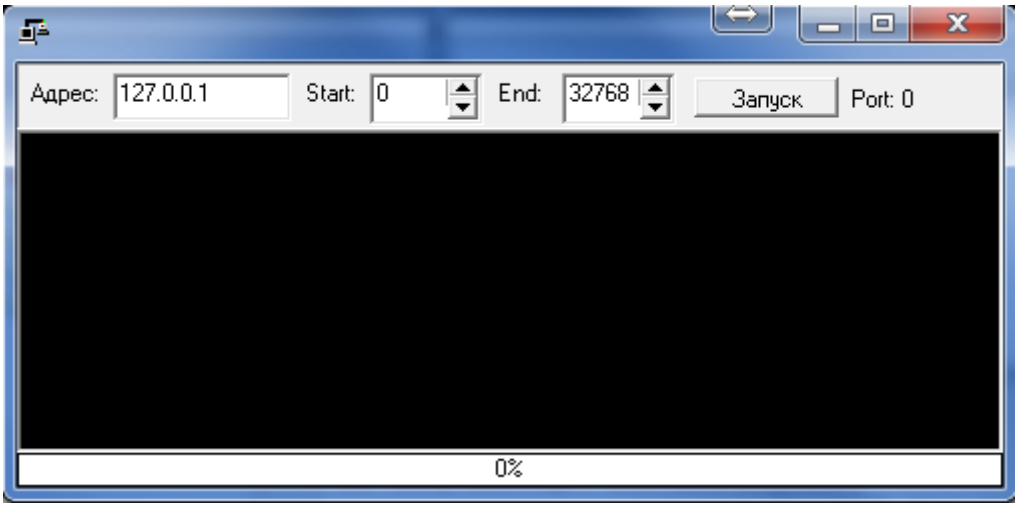

Рисунок 3.1 – Головне вікно додатку. Етап проектування.

При розробці програмного продукту було використано компоненти наступних класів:

- TPanel;
- TLabel;
- TEdit;
- TSpeedButton;
- TGauge;
- TListBox;
- TspinEdit.

3.2 Розробка основних функцій

Для забезпечення можливості підключення до сервера безлічі клієнтів сервер на кожне з'єднання запускає окремий потік.

```
program WinSock_Server;
// Сокети працюють в блокуючому режимі.
// На кожне з'єднання створюється окремий потік.
{$ APPTYPE CONSOLE}
uses
SysUtils,
Winsock,
Windows;
var
vWSAData: TWSAData;
vListenSocket, vSocket: TSocket;
vSockAddr: TSockAddr;
trId: THandle;
const
cPort = word (33);
cSigExit = 'q';
// Процедура окремого потоку для кожного клієнта.
procedure SocketThread;
var SockName: TSockAddr;
aBuf: array of char;
vBuf: string;
vSize: integer;
s: TSocket;
BufSize: integer;
```

```
begin
s: = vSocket;
if s = INVALID_SOCKET then exit;
vSize: = SizeOf (TSockAddr);
getpeername (s, SockName, vSize);
Writeln (format ( 'Client accepted, remote address [% s].', [Inet_ntoa (SockName.sin_addr)]));
// Визначаємо розмір буфера читання для сокета
vSize: = sizeOf (BufSize);
getsockopt (s, SOL_SOCKET, SO_RCVBUF, PChar (@
BufSize), vSize);
writeln (format ( 'Receive buffer size [% d]', [BufSize]));
SetLength (aBuf, BufSize);
repeat
// Отримуємо дані. Процедура працює в блокує режимі,
// таким чином наступний рядок коду не отримає управління,
// поки не надійдуть дані від клієнта.
vSize: = recv (s, aBuf [0], BufSize, 0);
if vSize <= 0 then Break;
SetLength (vBuf, vSize);
lstrcpyn (@vBuf [1], @ aBuf [0], vSize);
writeln (format ( 'Received from cleint:% s', [vBuf]));
until vBuf = 'q';
Writeln (format ( 'Client disconnected, remote address [% s].', [Inet_ntoa (SockName.sin_addr)]));
SetLength (aBuf, 0);
closesocket (s);
end;
begin
Writeln ( 'Starting application ...');
```

```
// Оголошуємо, що програма буде використовувати Windows Sockets.
if WSAStartup ($ 101, vWSAData) <> 0 then Halt (1);
Writeln ( 'Using Windows Sockets.');
// Створюємо прослуховуючий сокет.
vListenSocket: = socket (AF_INET, SOCK_STREAM, IPPROTO_IP);
Writeln (format ( 'Creating socket on port [% d].', [CPort]));
if vListenSocket = INVALID_SOCKET then Halt (1);
FillChar (vSockAddr, SizeOf (TSockAddr), 0);
vSockAddr.sin_family: = AF_INET;
vSockAddr.sin_port: = htons (cPort);
vSockAddr.sin_addr.S_addr: = INADDR_ANY;
Writeln ( 'Binding socket ...');
// Прив'язуємо адресу і порт до сокету.
if bind (vListenSocket, vSockAddr, SizeOf (TSockAddr)) <> 0
then Halt (1);
// Починаємо прослуховувати.
if listen (vListenSocket, SOMAXCONN) <> 0
then Halt (1);
Writeln ( 'Socket status: listening.');
repeat
// Очікуємо підключення.
vSocket: = accept (vListenSocket, nil, nil);
// Клієнт підключився, запускаємо новий процес на з'єднання.
CreateThread (nil, 0, @ SocketThread, 0,0, trId);
until false;
closesocket (vListenSocket);
WSACleanup;
end.
```

Використана функція getpeername (), яка повертає інформацію про канал, асоційоване з сокетом. Вона була використана для отримання інформації про IP-адресу підключеного клієнта.

Вихідний код клієнта який було розроблено для проведення тестів наведено нижче:

```
program WinSock_Client;
{$ APPTYPE CONSOLE}

uses
SysUtils,
winsock;
const
cPort = 33;
cSigExit = 'q';
var
vWSAData: TWSAData;
vSocket: TSocket;
vSockAddr: TSockAddr;
buf: string;
begin
if WSAStartup ($ 101, vWSAData) <> 0 then Halt (1);
vSocket: = socket (AF_INET, SOCK_STREAM, IPPROTO_IP);
if vSocket = INVALID_SOCKET then Halt (1);
FillChar (vSockAddr, SizeOf (TSockAddr), 0);
vSockAddr.sin_family: = AF_INET;
vSockAddr.sin_port: = htons (cPort);
vSockAddr.sin_addr.S_addr: = inet_addr ( '127.0.0.1');
if connect (vSocket, vSockAddr, SizeOf (TSockAddr)) = SOCKET_ERROR
then Halt (1);
repeat
```

```
Readln (buf);
if send (vSocket, buf [1], Length (buf), 0) = SOCKET_ERROR then Break;
until buf = cSigExit;
closesocket (vSocket);
WSACleanup;
end.
```

Для того щоб перевести сокет в неблокуючий режим, використовується функція ioctlsocket (...), що дозволяє контролювати режими роботи сокета.

Оскільки тепер функція recv сервера буде повертати управління відразу, незалежно від наявності даних в буфері сокета, то тепер потрібен механізм, що дозволяє визначати будь-які події, що відбуваються з сокетом. Для цього існує кілька механізмів. Перший, який розглянемо, це використання функції select (...).

function select (nfds: Integer; readfds, writefds, exceptfds: PFDSet; timeout: PTimeVal): Longint; stdcall; -Ця функція дозволяє контролювати стан набору сокетов.

Аргумент nfds ігнорується і залишений тільки для сумісності. Має дорівнювати 0. readfs, writefds, exceptfds - покажчики на набори сокетов, для яких потрібно контролювати стан читання, відправки даних і помилок відповідно. Набори зберігаються в структурі PFDSet, управління якою здійснюється спеціальними макросами, описаними в winsock.pas:

procedure FS_ZERO (var FDSet: TFDSet) - обнуляє структуру, встановлює кількість контрольованих сокетів в 0;

procedure FD_SET (Socket: TSocket; var FDSet: TFDSet) - додає вказаний сокет в структуру;

procedure FD_CLR (Socket: TSocket; var FDSet: TFDSet) - видаляє вказаний сокет зі структури;

function FD_ISSET (Socket: TSocket; var FDSet: TFDSet): Boolean - повертає true, якщо вказаний сокет є членом зазначеної структури.

Аргумент timeout є посиланням на структуру типу PTimeVal, в якій можна вказати час очікування спрацьовування функції select. У разі зазначення в якості значення часу затримки 0 або nil в якості аргументу timeout функція select буде чекати нескінченно, як при виконанні операції в блокуючому режимі.

Як видно з опису, функція може стежити відразу за декількома сокетами, таким чином, тепер ми можемо або також запускати окремий процес на кожен відкритий сокет, який буде стежити за конкретним сокетом, або створити один процес, який буде стежити за всіма відкритими сокетами. У першому випадку будуть потрібні лише незначні зміни процедури SocketThread, у другому випадку будуть потрібні досить значні зміни.

Розглянемо логіку роботи в другому випадку, так як перший випадок зрозумілий.

Для початку в основній частині програми необхідно перевести новостворений сокет в неблокуючий режим:

arg: = 1;

ioctlsocket (Socket, FIONBIO, arg);

Перед закриттям сокета його необхідно буде повернути в блокуючий режим:

arg: = 0;

ioctlsocket (Socket, FIONBIO, arg);

Потім необхідно реалізувати можливість збереження кожного сокета в деякий масив SockArray. Далі, потрібно забезпечити, щоб потік, який буде займатися обробкою клієнтів, запускався тільки один раз. Це зробити нескладно, знаючи, що процедура CreateThread повертає в змінну посилання на новостворений потік, яку і потрібно використовувати для перевірки, чи був створений потік. У потоці обробки потрібно внести наступні зміни для використання функції select: нові використовувані змінні wfds: TFDSet; i: integer; tv: Ttimeval;

Потім визначимо основний цикл, в якому будемо обробляти дані:

```
Repeat
...
until connum> 0;
```

У цьому циклі насамперед необхідно сформувати структуру wfds, що містить набір контрольованих сокетів. Для цього переносимо туди сокети з масиву SockArray:

```
FD_ZERO (wfds);
for i: = 1 to connum do
begin
FD_SET (sock [i], wfds);
end;
```

Далі, вказуємо в структурі tv час затримки для функції select:

```
tv.tv_sec: = 5;
tv.tv_usec: = 0;
```

Тепер можна викликати функцію select (тому що стежимо тільки за прийомом даних, то в якості writefds, exceptfds вказуємо nil):

```
select (0, @ wfds, nil, nil, @ tv);
```

Тепер, коли функція select поверне управління в змінної wfds, будемо мати набір сокетов, для яких необхідно провести читання, і можемо обробити дані, що надійшли:

```
if wfds.fd_count = 0 then continue;
for i: = 0 to wfds.fd_count-1 do
begin
vSocket: = wfds.fd_array [i];
// Обробка даних, що надійшли з сокета vSocket.
end;
```

Умова виходу з основного циклу - відсутність відкритих сокетів в масиві SockArray.

Сокет потрапить в обробку select і при настанні події розриву зв'язку (для обробки можна використовувати те, що кількість прийнятих байт функцією recv дорівнюватиме нулю, а також функцію WSAGetLastError).

В поточній реалізації, при створенні нового сокета, він не потрапить в обробку select, поки не буде встановлено функцією FS_SET, що, природно, не пройде в блокуючому режимі, поки select не поверне управління за подією з одним з відслідковуваних клієнтів . Установка значення timeout гарантує, що сокет потрапить в обробку незалежно від стану інших портів.

Якщо в процесі спілкування з клієнтом серверу потрібно чимало часу на підготовку і обробку даних, то, звичайно, слід створити різні потоки, щоб іншим клієнтам не довелося чекати завершення обробки, а працювати паралелльно. В іншому випадку, якщо обробка мінімальна, то створення великої кількості процесів вплине в гіршу сторону на загальну продуктивність системи.

Крім функції select використано ще два методи роботи з асинхронними сокетами:

function WSAAsyncSelect (s: TSocket; HWindow: HWND; wMsg: u_int; lEvent: Longint): Integer; stdcall;

Ця функція пов'язує сокет з отриманням повідомлень вікна. При виклику цієї функції, повідомлення про з'єднання, читання / запис даних в сокет і закриття сокета можна обробляти в функції обробки повідомлень вікна.

```
const
WM_MYSOCKET = WM_USER + 1;
...
type
TForm1 = class (TForm)
...
private
procedure Socket_Proc (var Msg: TMessage); message WM_MYSOCKET;
...
```

```
WSAAsyncSelect (vSocket, Form1.Handle, WM_
MYSOCKET, FD_ACCEPT + FD_READ);
....
procedure TForm1.Socket_Proc (var Msg: TMessage);
begin
if ((Msg.Msg = WM_MYSOCKET)
and (Msg.lParam = FD_ACCEPT))
then ShowMessage ( 'Connected');
end;
```

Файл winsock.pas не імпортує відповідні функції, в результаті чого можливостей подій важко використати. Тому розроблено власний імпорт необхідних процедур:

```
function WSAEventSelect (s: TSocket; Event: THandle; lEvent: Longint):
integer; stdcall;
external 'ws2_32.dll' name 'WSAEventSelect';
function WSAWaitForMultipleEvents (nCount: DWORD; lpHandles:
PWOHandleArray;
bWaitAll: BOOL; dwMilliseconds: DWORD; fAlertable: BOOL): integer;
stdcall;
external 'ws2_32.dll' name 'WSAWaitForMultipleEvents';
function WSACreateEvent: THandle; stdcall;
external 'ws2_32.dll' name 'WSACreateEvent';
function WSAResetEvent (Event: THandle): BOOL; stdcall;
external 'ws2_32.dll' name 'WSAResetEvent';
function WSAEnumNetworkEvents (const s: TSocket;
const Event: THandle; lpNetworkEvents: LPWSANetworkEvents): longint;
stdcall; far;
external 'ws2_32.dll' name 'WSAEnumNetworkEvents';
function WSACloseEvent (Event: THandle): integer;
stdcall; external 'ws2_32.dll' name 'WSACloseEvent';
```

Також нам потрібно опис структури WSANetworkEvents

```
const
fd_max_eventS = 10;
type
TWSANetworkEvents = record
lNetworkEvents: LongInt;
iErrorCode: Array [0..fd_max_eventS-1] of Integer;
end;
PWSANetworkEvents = ^ TWSANetworkEvents;
LPWSANetworkEvents = PWSANetworkEvents;
```

Принцип роботи з цим набором функцій полягає в створенні спеціального об'єкта типу Event, потім зв'язування цієї події з сокетом за допомогою функції WSAEventSelect, в якій також вказується набір відслідковуваних станів сокета. Один сокет може бути пов'язаний тільки з одним об'єктом типу Event.

Потім, в циклі обробки організуємо очікування надходження події від сокета; це реалізується за допомогою API функцій WaitForSingleObject - для очікування однієї події, або WaitForMultipleObjects - для очікування набору подій. При настанні події функція повертає управління. Для однозначної ідентифікації, від якого сокета прийшло повідомлення, в зв'язку з чим використовується функція WSAEnumNetworkEvents, що повертає структуру типу TWSANetworkEvents.

```
var
FEvent: THandle;
// Створюємо серверний сокет
...
FEventClose: = WSACreateEvent;
WSAEventSelect (Socket, FEvent, FD_CLOSE + FD_READ);
repeat
WaitForSingleObject (FEvent, INFINITE);
WSAEnumNetworkEvents (FSocket, FEvent, @ NI);
```

```
case NI.lNetworkEvents of
FD_Close: break;
FD_Read: begin
ReceiveData;
end;
end;
WSAResetEvent (FEventClose);
Until false;
WSACloseEvent (FEventClose);
```

Розглянемо відправку даних - відправка даних є, по суті, постановка порції даних в чергу. Не можна керувати окремими пакетами, більш того, дані, що потрапили в буфер, відправляються не відразу, а можуть накопичуватися для відправки в подальшому одним пакетом.Таким чином, послідовний виклик

```
send (vSocket, @ buf1, Length (buf1), 0);
send (vSocket, @ buf2, Length (buf2), 0);
```

фактично буде ідентичний кільком особам send з об'єднаним буфером buf1 + buf2.

Таким чином, при прийомі даних, хоча послано дві порції даних, буде отримано одну. Зовсім інший випадок, коли послано порцію даних, більшу, ніж буфер сокета, тоді функція send відправить тільки дані з вказаного буфера, рівно стільки, скільки дозволить розмір буфера сокета. Для того щоб відстежити таку ситуацію і відправити необроблену частину буфера, потрібно скористатися тим, що функція send повертає кількість фактично відісланих даних. Функцію send потрібно запускати в циклі, умовою завершення якого буде повна відправка всього буфера.

Таким чином, при читанні з сокета даних, можна спостерігати фрагментацію відправки даних. Такі ситуації повинна обробляти програма. Вирішено цю проблему за допомогою додаванням сигнатури ознаки кінця блоку даних.

На початку кожного пакета додається 32-бітове число, що визначає довжину порції даних в байтах. Таким чином, приймаюча частина, знаючи розмір кожного блоку, може розпізнати фрагментацію.

3.3 Тестування та верифікація програмного засобу

Тестування проводилось на компютері: Intel Core i3-6006U (2.0 ГГц) / RAM 4 ГБ / HDD 500 ГБ / Intel HD Graphics 520 / без ОД / LAN / Wi-Fi / Bluetooth. При першому запуску додатку користувачеві буде відображена фома, що на рисунку 3.2.

Для тестування було проведене сканування всіх можлдивих портів для локального та віддаленого вузлів. Проведено виконання тестових прикладів з спеціально відкритими портами.

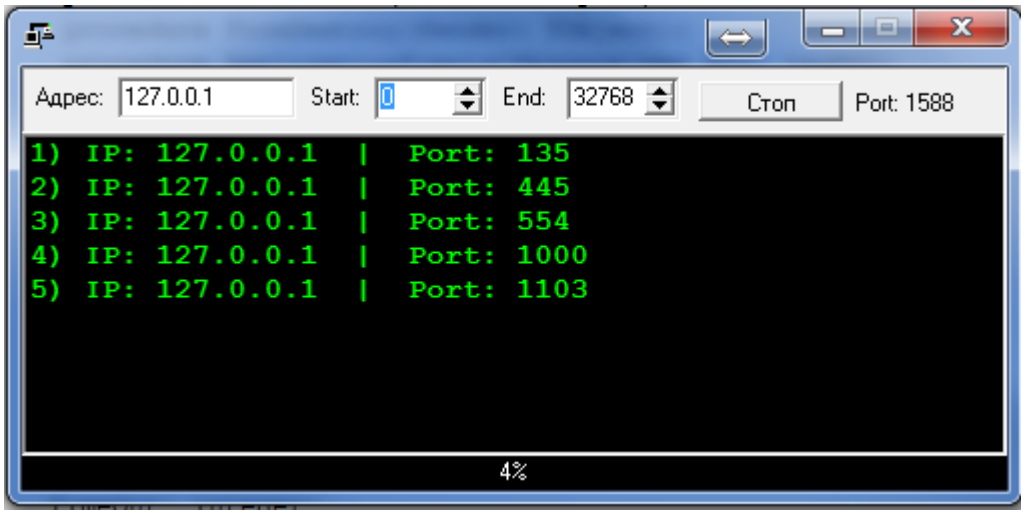

Рисунок 3.2 – Головне вікно додатку. Сканування локального компютера. Етап тестування.

Для тестування програмного засобу при скануванні віддалених вузлів було проведене тестування з використанням віддаленого вузла за адресою 192.168.2.2, як показано на рисунку 3.3.

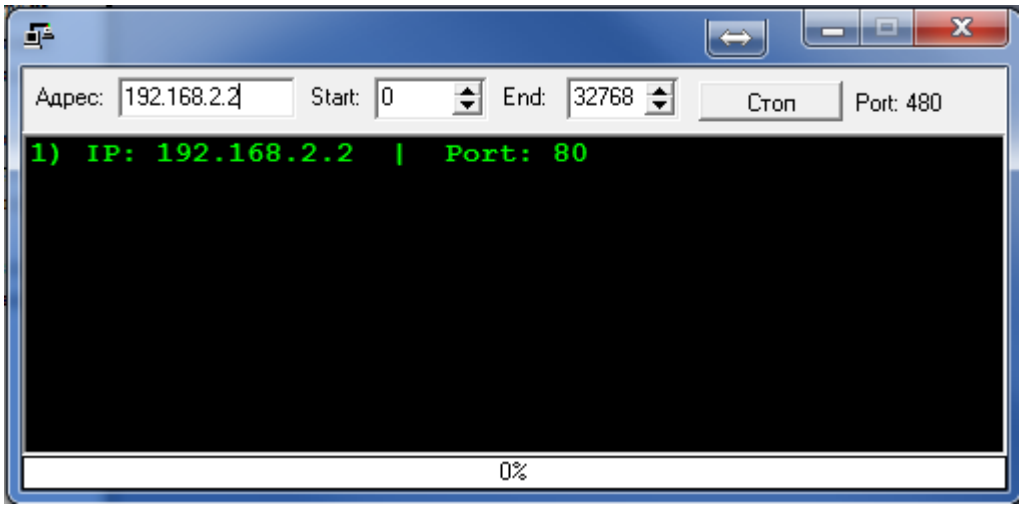

Рисунок 3.3 – Головне вікно додатку. Сканування віддаленоговузла. Етап тестування.

При проведені тестування не було виявлено проблем із сумсністю та роботою брендмауера/файервола. Програмний продукт в результаті виконання тест плану пройшов перевірку функціональності. В результаті перевірки серйозних проблем не було виявлено.

ВИСНОВКИ

Тестування проводилось на компютері: Intel Core i3-6006U (2.0 ГГц) / RAM 4 ГБ / HDD 500 ГБ / Intel HD Graphics 520 / без ОД / LAN / Wi-Fi / Bluetooth.

Для тестування було проведене сканування всіх можлдивих портів для локального та віддаленого вузлів. Проведено виконання тестових прикладів з спеціально відкритими портами.

При проведені тестування не було виявлено проблем із сумсністю та роботою брендмауера/файервола. Програмний продукт в результаті виконання тест плану пройшов перевірку функціональності. В результаті перевірки серйозних проблем не було виявлено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Семенов Ю. А. Телекоммуникационные технологии. Интернет-университет информационных технологий [Электронный ресурс]. – Режим доступа: http://book.itep.ru

2. Олифер В. Г. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов: 3-е изд. / В. Г Олифер., Н. А. Олифер // – СПб.: Питер, 2006.

3. Зайченко Ю. П. Анализ и оптимизация характеристик сетей MPLS по заданным показателям качества / Ю. П. Зайченко, Ахмед А. М. Шарадка // Вісник національного технічного університету України КПІ сер. Інформатика управління та обчислювальна техніка. Вип. 43. – 113–123 с.

4. Будылдина Н. В. Разработка программного обеспечения для оптимизации мультисервисных сетей / Н. В. Будылдина., П. А. Коновалов // Открытое образование, июнь 2006. – 58 с.

5. Зайцев Д. А. Моделирование телекоммуникационных сетей в системе NS. / Д. А. Зайцев, Т. Н. Шинкарчук // Наукові праці ОНАЗ ім. О. С. Попова. – 2006.– № 2.

6. Кучерявый Е.А. Управление трафиком и качество обслуживания в сети Интернет / Е.А. Кучерявый // – М.: Наука и Техника. – 2007. – 336 с.

7. Panwar Li. Y. S. On the Performance of MPLS TE Queues for QoS Routing // Panwar Li. Y. Liu C.J. Simulation series. – 2004. – Vol. 36; part 3. – P. 170–174.

8. Аткинсон Л. Mysql. Библиотека профессионала – М Энергоатомиздат, 2002 – 496 с.

9. Дейт К. Руководство по реляционной СУБД DB2 – М.: Финансы и статистика, 1988 – 320 с.

10. Мейер М.Теория реляционных баз данных – М.: Мир, 1987 – 608 с.

11. Семантична модель: База даних [Електронний ресурс] Режим доступу: http://citforum.ru/database/advanced_intro/27.shtml.

12. Риккарди Г. Системы баз данных. Теория и практика использования в Internet – М.:Вильямс, 2001 – 240с.

13. ДейтК. Дж. Введение в системы баз даннях – СПб: Изд-во "Пітер", 2005 – 1315с.

14. Роберт І.В. Сучасні інформаційні технології в освіті: дидактичні проблеми, перспективи використання — М.: Школа-Пресс, 1994— 205с.

15. Розділ 2.Основи UML – діаграм: [Електронний ресурс] Режим доступу:https://docs.kde.org/trunk4/uk/kdesdk/umbrello/uml-basics.html

16. Мюллер Р.Дж. Базы данных и UML. Проектирование / Перевод. с англ. Е. Молодцова. – М.: Издательство “Лори”,2002 – 432с.

17. Острей О.Р. Діаграми класів UML як засіб моделювання інформаційної системи моніторингу освіти / М.: - 2008. № 2. – С.85-89.

18. Електронна енциклопедія Вікіпедія: JSON[Електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/JSON

19. Компонентне або модульне тестування: [Електронний ресурс] Режим доступу: http://www.protesting.ru/testing/levels/component.html

20. Електронна енциклопедія Вікіпедія: Модульне тестування [Електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/Модульне

21. Електронна енциклопедія Вікіпедія: Список кодів стану HTTP [Електронний ресурс] Режим доступу:https://uk.wikipedia.org/wiki/список

22. Шапошников І. Web-сайт своїми руками./ І. Шапошников – СПб: Изд-во "Пітер", 2002 – 390с.

23. Гаевский А. Ю. Самоучитель по созданию Web-страниц: HTML, JavaScript, Dynamic HTML / А. Ю. Гаевский, В. А.Романовский. — К.: А.С.К., 2002 — 472с.

24. Методичні вказівки до написання техніко – економічного розділу для дипломних проектів на здобуття освітньо – кваліфікаційного рівня «Бакалавр» напряму підготовки 6.050102 «Комп’ютерна інженерія» / І.Р. Паздрій. – Тернопіль: ТНЕУ, 2015. – 36с.

25. Методичні рекомендації до виконання дипломного проекту з освітньо – кваліфікаційного рівня «Бакалавр» напряму підготовки 6.050102 «Комп'ютерна інженерія» фахового спрямування «Комп'ютерні системи та мережі» / Л.О. Дубчак, О.М. Березький, Р.Б Трембач, Г.М. Мельник, Ю.М. Батько, С.В. Івасьєв / Під ред. О.М. Березького. – Тернопіль: ТНЕУ, 2016.- 60с.