МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Кафедра кібербезпеки та програмного забезпечення

На правах рукопису

Хуторний Дмитро Олексійович

Програмне забезпечення системи Network Performance Monitoring

Спеціальність: 123 «Комп'ютерна інженерія»

Освітній ступінь: бакалавр

Науковий керівник:

Дрєєв Олександр Миколайович _____

(підпис) (дата)

кандидат технічних наук, доцент

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

_____ О.А. Смірнов

(підпис) ПІБ

«____»_____2021 р.

Кропивницький – 2021

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *123 Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« 11 » січня 2021 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Хуторному Дмитру Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи Network Performance Monitoring*

керівник роботи *Дрєєв Олександр Миколайович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 204-02 від 28.12.2020 року

2. Строк подання студентом роботи до захисту *22.05.2021 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення системи Network Performance Monitoring*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання « 11 » січня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2021 р.	
3.	Розробка моделі компонента	20.03.2021 р.	
4.	Розробка структур даних	25.03.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2021 р.	
6.	Програмування алгоритмів	10.04.2021 р.	
7.	Оформлення ПЗ	17.04.2021 р.	
8.	Попередній захист роботи	14.05.2021 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Хуторний Д.О. Програмне забезпечення системи Network Performance Monitoring. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній кваліфікаційній бакалаврській розроблено програмне забезпечення, яке призначено для системи Network Performance Monitoring.

Метою розробки є програмне забезпечення системи Network Performance Monitoring.

Результат роботи – програмна реалізація системи Network Performance Monitoring.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows XP/Vista/7/8/10.

Програму розроблено в середовищі Delphi 10.4 Sydney.

Ключові слова: комп'ютерна інженерія, Network Performance Monitoring

ABSTRACT

Khutornyi D.O. Network Performance Monitoring software. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

In this bachelor's qualification the software which is intended for the Network Performance Monitoring system is developed.

The purpose of the development is the network performance software Monitoring system.

The result is a software implementation of the Network Performance Monitoring system.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

Developed user-friendly interface. Instructions for working with software are given.

The program can be used on an IBM PC with Windows XP / Vista / 7/8/10.

The program is developed in the environment of Delphi 10.4 Sydney.

Keywords: computer engineering, Network Performance Monitoring

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ІТ	–	інформаційні технології
ПЗ	–	програмне забезпечення
QoS	–	механізми контролю й керування якістю

ВСТУП

Актуальність теми. Network Performance Monitoring (NPM), також називаний керуванням продуктивністю мережі, визначається як практика виміру, аналізу й оптимізації якості обслуговування в мережі з погляду користувача. Ефективні монітори продуктивності мережі характеризують і повідомляють про відповідні показники, пов'язані з IT-послугами й допоміжними ресурсами. Вони також надають компоненти для усунення неполадок мережі, застосунків або аномалій безпеки, а також можливості, призначені для поліпшення загального надання послуг і сприйнятливості кінцевих крапок.

Інструменти використовують різні типи даних, включаючи дані мережного потоку, пакетні дані й метрики мережної інфраструктури. Володіючи вичерпними даними в реальному часі й криміналістичними даними, а також чудовими аналітичними можливостями, мережні адміністратори можуть управляти повсякденними операціями й відслідковувати тенденції, оптимізуючи продуктивність і зводячи до мінімуму ризики безпеки.

Зростаючі проблеми мережного моніторингу пов'язані з тим, що розвивається гібридним IT-середовищем у цілому й активами, розміщеними в хмарі, зокрема, з такими абстракціями ресурсів, як віртуалізація віртуальних серверів і програмно-обумовлені мережі (SDN). NPMD може допомогти. З обліком того, що функції керування, що раніше виконувалися традиційними маршрутизаторами, серверами й міжмережевими екранами, тепер переходять на програмні розв'язки, методи моніторингу повинні були швидко адаптуватися до цих віртуалізованим функціям.

Застосунки також стають усе більш складними в міру збільшення кількості рівнів і варіантів їх розміщення. Більше не обмежуючись локальним розгортанням у минулому, застосунки тепер можна розділити між декількома хмарними або зовнішніми географічними місцями розташування, що частково

збігаються із традиційними локальними розгортаннями. Ця зростаюча складність привела до прогресивного й індивідуального підходу, який містить у собі більше аналітики великих даних, машинне навчання, варіанти програмного забезпечення для моніторингу продуктивності мережі й хмарні обчислення.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи Network Performance Monitoring.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем Network Performance Monitoring.

– Дослідження системи Network Performance Monitoring.

– Програмна реалізація системи Network Performance Monitoring.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі Network Performance Monitoring.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи Network Performance Monitoring, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Згідно із глобальним дослідженням стану мережі VIAVI Solutions за 2020 рік, більше половини мережних фахівців указали, що єдиною найбільшою проблемою при усуненні неполадок є визначення причини проблеми. По суті, зіштовхнувшись із проблемами продуктивності, мережні групи не знають, із чого почати усунення неполадок – будь то мережа, програмне забезпечення, застосунок, мобільний застосунок, клієнт або система. Ці проблеми ускладнюються при керуванні продуктивністю мережі в хмарних або віддалених середовищах.

Enterprise NPM також являє собою складну проблему поширення пристроїв, викликану Інтернетом речей (IoT), що тільки збільшує дилему. Централізація NPM може бути ефективною стратегією для відомості до мінімуму розрізненості, яка іноді приводить до неповного вирішення проблем і відсутності загальної видимості всієї мережі, необхідної для ефективних методів усунення неполадок.

1.2 Область застосування

Областю застосування є усунення неполадок у мережі. Симптоми легко розпізнати. Повільні з'єднання, збої й перебої в роботі – це одні із проблем, про яких найчастіше повідомляють користувачі. Переваги полягають у швидкому визначенні правильної основної причини, пов'язаної із симптомами, щоб можна було почати ефективні коригувальні й профілактичні дії.

По даним Forrester Research, на розв'язок майже однієї третини всіх проблем із продуктивністю мережі йде більше місяця або вони ніколи не

вирішуються. Ці невирішені або проблеми, що повільно виправляються, безсумнівно, будуть повторюватися без ефективної стратегії вирішення проблем.

Розв'язок для моніторингу продуктивності мережі часто дає можливість краще візуалізувати показники й іншу важливу інформацію за допомогою панелей моніторингу, дисплеїв і графіків. Можливості діагностики й аналітики більшості розв'язків можуть бути використані для застосування автоматичного аналізу усунення неполадок до будь-якої виникаючої проблеми продуктивності або безпеки. Оповіщення, засновані на певних рівнях продуктивності, є ще одним важливим інструментом, що підвищують поінформованість про потенційні проблеми до того, як вони досягнуть критичного рівня зниження продуктивності.

Усунення проблем може бути значно прискорене за допомогою розв'язків, які відслідковують, фіксують і забезпечують довгострокове зберігання всіх пакетів, традиційного потоку, системних журналів, SNMP, IP-адрес і MAC-адрес мережних пристроїв і інтерфейсів, розмов і транзакцій. Збереження цієї важливої інформації й розширена аналітика дозволяють IT-командам швидко переглядати масивні топології мережі, відслідковуючи масивні обсяги трафіка після події, і знаходити конкретний час, коли відбувся епізод, незалежно від того, чи був він пов'язаний із проблемою обслуговування або порушенням безпеки. Не потрібно чекати, поки проблема повториться. Після виявлення аномалію можна переглянути в контексті інших дій у міру її виникнення. Думайте про це як про «постійно включену» систему відеоспостереження для вашої мережі.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи Network Performance Monitoring, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

VIAVI (NPMD)

Інструменти моніторингу й діагностики продуктивності мережі VIAVI (NPMD) пропонують можливість витягати мережні дані й застосовувати глибокий аналіз для оптимізації IT-ресурсів у режимі реального часу або після подій.

Занадто багато вказівок на команду мережі й недостатньо часу, щоб прочитати скарги й випередити їх? Журнали? Звіти? Час відгуку? Сама більша проблема при усуненні неполадок продуктивності мережі часто полягає в тому, щоб знати, із чого почати пошук фактичного джерела проблеми із продуктивністю. Ця проблема може збільшитися, якщо стандартні мережі стануть абстрактними й децентралізованими, будь то IoT, SD-WAN або складна топологія хмарних технологій SaaS.

У сьогоднішньому світі віддаленої роботи, де багато IT-відділів обмежені дистанційним доступом до користувачів і їх кінцевим крапкам, потреба в комплексному моніторингу інфраструктури й аналізі стану віддалених кінцевих користувачів як ніколи гостріше.

Інструменти NPM від VIAVI завчасно виявляють найбільш серйозні мережні проблеми за допомогою оцінки досвіду кінцевого користувача, діагностики проблем і готових робочих процесів, що ведуть до розв'язку IT-фахівців. Увесь інтелект Observer підтримується повними, незмінними пакетними й потоковими даними, надаючи провідні дані для детального дослідження безпеки, спостерігаючи за обміном даними по протоколу керування передачею з унікальними IP-адресами, зв'язками між MAC-адресами й

ідентифікаторами користувачів, потужним аналізом мережних підключень і аналізом першопричин для оптимізована робота мережі й застосунків.

Observer v18 і Gigastor Gen 4

Тепер, з випуском версії Observer v18, пакетні дані й дані розширеного потоку тепер співіснують в Observer Apex. Це означає, що всі рівні досвіду одержують доступ до різних рівнів видимості IT, використовуючи свої кращі джерела даних для виміру Qos, визначення базових параметрів, планування ємності й інших варіантів використання продуктивності мережі.

З Gigastor Gen 4 VIAVI також забезпечує найшвидшу незалежно підтверджену швидкість захвата даних по проводах зі швидкістю 60 Гбіт/с в одному простому в установці пристрої, що дозволяє вам масштабуватися відповідно до вимог сучасного світу, у якому усе більше й більше підключень.

Observer забезпечує всебічну видимість стану й аналіз критично важливих IT-ресурсів, будь те в ядрі центру обробки даних, на границі мережі або в хмарі. Надаючи інформацію про мережні й прикладні продукти з погляду кінцевого користувача, Observer підтримує мережні й операційні групи, щоб забезпечити максимальну ефективність IT-послуг і центрів обробки даних, від порту до порту, у трьох критичних сценаріях використання:

– Керування новими ініціативами й повсякденними операціями.

– Зниження ризику від запланованої й несподіваної події.

– Вирішення проблем із продуктивністю й безпекою.

Розв'язок для моніторингу продуктивності мережі від VIAVI служить відправною крапкою для усунення неполадок у роботі сервісних аномалій, втрати пакетів, керування мережними ресурсами й допомоги в розслідуванні проблемних інцидентів або підтверджених порушень безпеки. Ця відправна крапка гарантує, що IT-групи можуть швидко одержати доступ до відповідних до застосунків, мережі, бази даних, інфраструктурі й експлуатаційним показникам користувачів для різних критично важливих сценаріїв продуктивності мережі й

мережної безпеки. Це означає менше часу на усунення неполадок і більше часу на виконання цифрової трансформації в Інтернеті для вашого бізнесу.

Solarwinds Network Performance Monitor

Solarwinds Network Performance Monitor – економічний, простий і потужний інструмент, що дозволяє в режимі реального часу відслідковувати продуктивність мережі.

Продукт Solarwinds Network Performance Monitor швидко виявляє, діагностує й сприяє у вирішенні проблем швидкодії мережі, перше ніж виникнуть простої в роботі. Крім того, завдяки динамічним картам топології мережі й автоматичному виявленню її компонентів адміністратори можуть легко масштабувати мережа й у міру її росту приводити у відповідність важливі процеси.

Solarwinds Network Performance Monitor контролює час відгуку, доступність і період працездатності маршрутизаторів, комутаторів і інших пристроїв з підтримкою SNMP.

Статистика продуктивності мережі відображається в реальному часі завдяки динамічним мережним картам. До складу розв'язку входять інструменти генерації повідомлень, звітів, а також керівництва й файли допомоги.

Ключові особливості:

– Автоматичне виявлення мережі. Network Performance Monitor дозволяє планувати процеси автоматичного сканування мережі, ідентифікувати нові мережні пристрої й забезпечувати моніторинг усього критично важливого встаткувань.

– Підтримка пристроїв від різних вендорів. Network Performance Monitor підтримує гетерогенні мережі й пристрою від провідних виробників устаткування.

– Моніторинг продуктивності й доступності мережі. Продукт відслідковує доступність і індикатори продуктивності мережних пристроїв і інтерфейсу, такі як навантаження на смугу пропущення, затримки, відповіді,

втрата пакетів, центральний процесор і пам'ять, по кожному встаткуванню з підтримкою SNMP і WMI.

– Інтелектуальні повідомлення. Network Performance Monitor дозволяє швидко конфігурувати оповіщення про події, умови й станах мережних пристроїв. При необхідності блокування повідомлень на основі залежностей і топології дозволяє адміністраторам одержувати оповіщення тільки по важливих проблемах мережі.

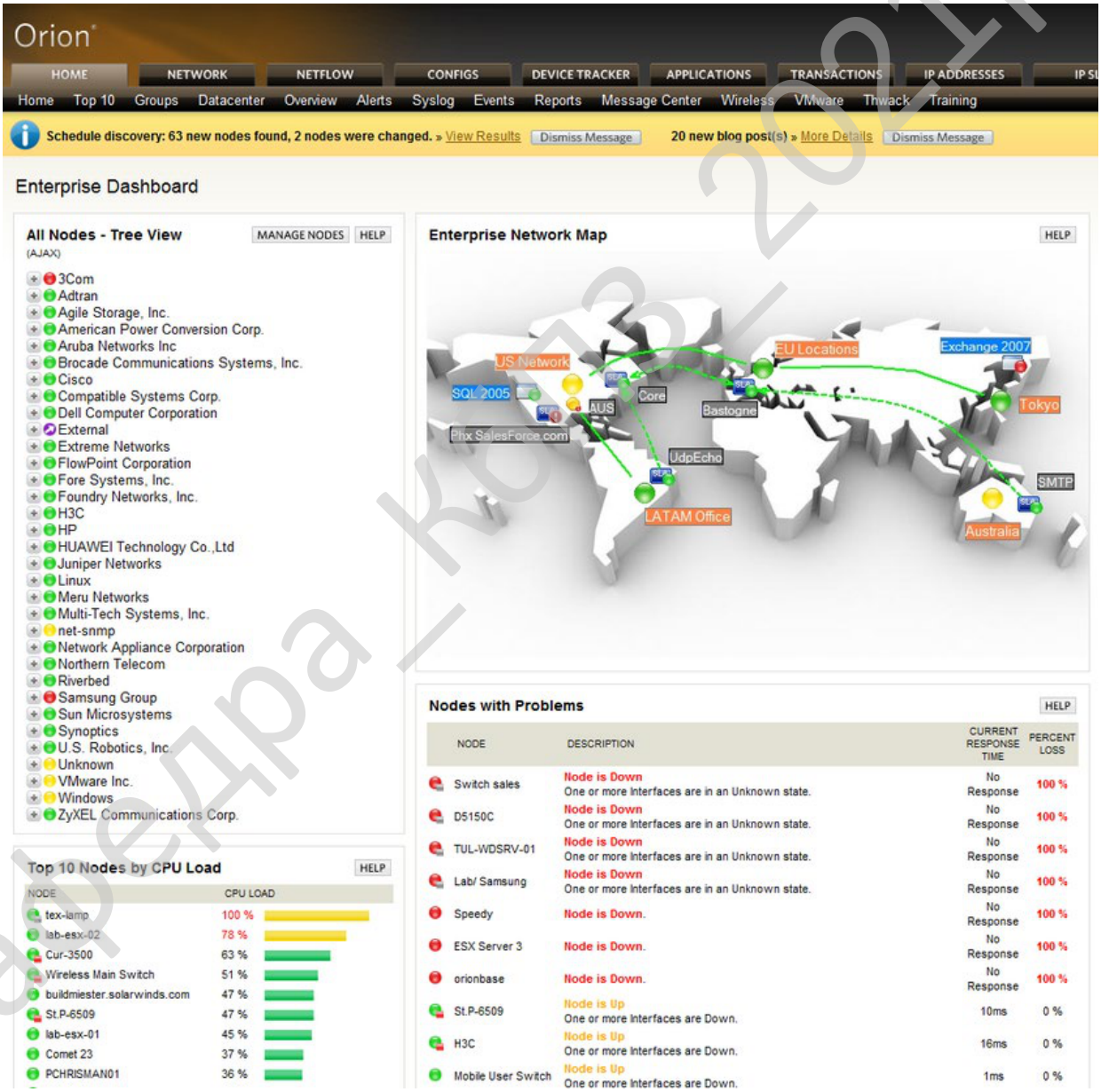

Рисунок 2.1 – Інтерфейс користувача Solarwinds Network Performance Monitor

– Швидке розгортання. Завантаження, установка й розгортання Network Performance Monitor здійснюються менш чому протягом години за три прості кроки.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для maCoS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на WkwebView API. Реалізація компонента Media Player для maCoS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

− Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за

допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на maCoS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL,

включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.
- Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.
- Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній

формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для maCoS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на WkwebView API.

– Реалізація компонента Media Player для maCoS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на кваліфікаційну бакалаврську роботу, реалізації підлягає програмне забезпечення, яке призначено для системи Network Performance Monitoring.

В процесі розробки кваліфікаційної бакалаврської роботи необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Три ключові можливості забезпечують ефективну стратегію моніторингу продуктивності мережі:

– Користувацький досвід: перша й найважливіша із цих можливостей – точне й своєчасне розуміння користувацького досвіду. Оскільки про 40% проблем уперше повідомляють користувачі, звіти про оцінку якості роботи кінцевого користувача є одними з найбільш інформативних і коштовних доступних показників продуктивності мережі. Удосконалений інструмент для підвищення продуктивності мережі може використовувати адаптивний інтелект для моніторингу стану мережі й на застосунок до збору інших деталей з погляду користувача. Це може спростити й пріоритезувати методи вирішення проблем, одночасно підвищуючи задоволеність і втримання клієнтів. Довідайтеся більше про нашу оцінку якості обслуговування кінцевих користувачів.

– Дані повної судової експертизи: Друга ключова можливість – це можливість приєднання до повних криміналістичних даних для розслідування. Це містить у собі записи розширеного потоку, що вказують історичний веб-трафік і мережний трафік, використання застосунків і пристроїв у мережі, а також дані на рівні пакетів з більш детальним рівнем інформації про конкретний файл і Url-адресі, які можна швидко викликати для відтворення й аналізу конкретних екземплярів. або викрійки. Ключовий момент, який слід за своїти, полягає в тому, що успішне усунення неполадок ґрунтується на прозорості кожної події й збереженні всіх транзакцій і цілих мережних розмов у вигляді фіксованих, а потім уніфікованих провідних даних. Відсутні події приводять до помилкового аналізу й зводять нанівець цінність даних, тому інструменти мережної криміналістики з великою ємністю зберігання й відкликання необхідні для

керування докладними й безперервними історичними шаблонами з метою оптимізації часу відгуку.

– Оптимізовані робочі процеси: Третя важлива можливість надійного розв'язку – це включення оптимізованих робочих процесів, призначених для швидкого й точного усунення розриву між ідентифікацією проблеми і її першопричиною. Ці розв'язки вимагають машинного навчання для пошуку, що попереджає, визначення й оцінки впливу подій на кінцевих користувачів, а також визначення масштабів і причин. Коли повідомляється про проблеми, командам потрібні готові робочі процеси, які оцінюють вплив проблем на кінцевих користувачів, допомагають знайти логічну причину проблеми й надають судові докази для її усунення.

У минулому мережні операції (NetOps) і операції безпеки (SecOps) залишалися відносно ізольованими друг від друга. Сьогодні все більше число організацій повністю об'єднали свою IT-безпека з керуванням продуктивністю мережі в одне співтовариство. Це має логічний сенс тільки тоді, коли проблеми, пов'язані з безпекою, є одними з найпоширеніших факторів, що сприяють виникненню проблем з мережними послугами. Ризик погроз безпеки може проявлятися в несподіваних подіях, які приведуть до дорогим простоїв або потенційно скомпрометованим даним. Цей ризик знижується за рахунок високоточної криміналістичної експертизи й передових можливостей розслідування, які прискорюють вживання відповідних заходів.

Комплексні розв'язки підвищують безпеку, пропонуючи докладну інформацію про трафік, що проходить по всій мережі, і базовій інфраструктурі, яка контролює потоки розмов. Завдяки такій же прозорості своєї мережі, команди SecOps і NetOps можуть говорити на одній мові, що сприяє розширенню співробітництва між двома традиційно ізольованими відділами. Зокрема, групи SecOps і NetOps можуть профілювати дії користувачів і пристроїв, швидко виявляючи й попереджаючи про можливу шахрайську поведінку. Крім того, документація про виявлення інформації важлива після завершення заходу для

очищення скомпрометованих активів і конкретної оцінки того, які інтелектуальні матеріали компанії могли бути скомпрометовані.

Переваги моніторингу продуктивності мережі

Технологічні інновації, стратегічна міграція в хмару й стимулювання цифрової трансформації – от деякі із загальних організаційних цілей, що вимагають від висококваліфікованих і гнучких IT-команд для збереження конкурентоспроможності, що приводить до проблем для багатьох IT-директорів при створенні належного балансу між бізнес-інноваціями й операційна досконаліть. Таким чином, переваги, які дає оптимізований розв'язок, включають комплексний підхід, що підтримує операційну перевагу IT-команд, що дозволяє вашій організації визволити ресурси для пошуку способів стимулювання цифрових інновацій і змін.

Цей багаторівневий підхід включає керування повсякденними операціями, зниження ризиків, пов'язаних із запланованими й незапланованими подіями, а також оптимізоване розслідування й вирішення проблем із продуктивністю.

Керування повсякденними операціями повністю забезпечується за допомогою гнучких панелей моніторингу й звітів, що забезпечують ситуаційну поінформованість у масштабах усього підприємства й видимість показників і тенденцій стану мережі в режимі реального часу. Ця поліпшена видимість також важлива для зниження ризиків змін і несподіваних подій.

Згідно з галузевими дослідженнями, середня вартість однієї години простою мережі становить 336 000 доларів. Це дозволяє залишатися на крок попереду потенційних проблем, включаючи всі, від людських помилок до проблем з конфігурацією й погроз безпеки, що необхідно для підтримки прибуткової роботи із задоволеними клієнтами.

У той час як інструменти моніторингу мережі з відкритим вихідним кодом, безумовно, можуть допомогти у виявленні першопричини, комплексний розв'язок для моніторингу продуктивності мережі, оснащене передовий криміналістикою й систематичними робочими процесами, може прискорити

вирішення проблем із продуктивністю навіть при суб'єктивних скаргах кінцевих користувачів.

Раніше адміністратори зосереджували на тому, як відслідковувати базову продуктивність мережі, але базового моніторингу вже недостатньо. Загальний упор на операційну перевагу має безліч аспектів, кожний з яких сприяє постійній доступності, продуктивності й безпеки мережі. Одним з важливих аспектів, що зачіпають кілька основних напрямків, є виявлення тенденцій.

При постійному моніторингу тенденції можуть діяти як дорожня карта, що веде до потенційного зниження продуктивності або погрозам безпеки. Ці тенденції також можна зрівняти з базовими показниками продуктивності для виявлення потенційно небезпечних умов, у тому числі тих, які виникають тільки спорадично або в години пікового навантаження.

NPM також може відігравати важливу роль у плануванні мережі й керуванні конфігурацією. Звіти про використання ресурсів і смуги пропущення доступні для підтримки функцій планування мережі, пов'язаних з ростом. Крім цієї коштовної повсякденної функціональності, неможливо переоцінити важливість забезпечення безпеки мереж і захисту даних. При постійному моніторингу трафіка потенційно небезпечні файли або дії можуть бути ідентифіковані в режимі реального часу й відповідним чином розширені.

Ключові особливості розв'язку NPM

Для ефективного моніторингу продуктивності, створення цінності, впровадження інновацій і забезпечення постійної безпеки необхідно враховувати кілька важливих функцій. У список критичних функцій, безумовно, входять можливості безперервного моніторингу, аналізу й візуалізації в реальному часі. Враховуючи складність сучасних мереж і постійно присутні погрози безпеки, будь-яка затримка, пов'язана з поінформованістю й реагуванням, може виявитися пошуку.

Віддалений моніторинг: кращі розв'язки для моніторингу продуктивності мережі також містять у собі можливості віддаленого моніторингу, які дозволяють

виконувати складні операції по збору й аналізу даних з різних місць, мобільних пристроїв і хмарних розгортань, а також необхідний інтелект, необхідний для автоматичного виявлення й планування нових пристроїв і встаткування.

Взаємодія з кінцевим користувачем: Ще одна важлива особливість цих систем – включення досвіду й перспективи кінцевого користувача в практику моніторингу. Кінцевий користувач є найбільш важливим і актуальним барометром продуктивності системи керування продуктивністю мережі, тому настійно рекомендується відслідковувати продуктивність кінцевих крапок і використовувати оцінку досвіду кінцевих користувачів. Точно так само ефективне налаштовування користувацького інтерфейсу у вигляді гнучких панелей моніторингу, інтуїтивно зрозумілої графіки й показників є важливими функціями для підвищення продуктивності й задоволеності IT, поряд з оптимізованим покриттям мережі й легенями для читання звітами. Входи, виходи й рівні попереджень повинні бути легко програмувальними й доступними.

Автоматизація. Поряд з поліпшеною видимістю й оперативністю на обох кінцях спектра, функції автоматизації важливі для діагностики й виправлення проблем без прямого контролю. Автоматизація стає усе більш коштовної через складність, властиву поширенню хмар, через експонентне збільшення шляхів підключення. Відомість до мінімуму кількості проблем, що вимагають прямого ручного реагування з боку IT-команди, може стати додатковим фактором підвищення ефективності й підвищення операційної ефективності.

Одержання правильного розв'язку для огляду мережі для вашої організації – це тільки початок. Визначення більш ефективного процесу моніторингу інфраструктури вимагає огляду, аналізу й розміщення пріоритетів, щоб переконатися, що цей коштовний ресурс використовується ефективно. Важливі розв'язки щодо того, що і як відслідковувати, і як часто є необхідними передумовами для визначення ваших ключових показників ефективності. Для установки рівнів попереджень системи необхідно додаткове планування й аналіз. Щоб зробити попередження значимими й запобігти інформаційному

перевантаженню, ці рівні повинні бути встановлені якнайближче до фактичного граничного значення тенденцій або подій, що впливають на сервіс. Ці міркування по налаштовуванню супроводжуються ролями й обов'язками персоналу, що визначають, хто буде відслідковувати, хто буде одержувати попередження і як буде здійснюватися доступ до інформації і її спільне використання.

Основні функції Network Performance Monitor:

– Автоматизоване обстеження мережних пристроїв.

– Можливість швидкої самостійної установки.

– Панель керування продуктивністю й станом застосунків.

– Глибокий аналіз пакетів і спеціальні сенсори.

– Створення, що налаштовуються динамічних карт для спостереження за станом мереж.

– Інтерактивні графіки для спостереження за продуктивністю мережних пристроїв.

– Моніторинг по протоколах RIP v2, OSPF v2 & v3, EIGRP, BGP.

– Інтуїтивний, що й гнучко налаштовується web-інтерфейс.

– Централізоване керування повідомленнями й нагадуваннями.

– Можливість установлювати динамічні базові лінії на основі статистики.

– Можливість угруповання мережних пристроїв по будь-яких категоріях.

– Що гнучко налаштовуються звіти по продуктивності й працездатності мереж.

– Моніторинг бездротових каналів і автономних крапок доступу біля провідних пристроїв.

– Моніторинг Snmp-пристроїв, збір докладних даних про Mib-Таблицях.

– Інтеграція з Microsoft Active Directory.

– Моніторинг і звітність по стану мережі Virtual Storage Area Network.

– Моніторинг складних складених широкомовних мереж.

– Підтримка керування компонентами Cisco Unified Computing System.

– Перегляд мережної статистики на мобільних пристроях iPhone, Blackberry, Android.

– Інтеграція з Microsoft System Center Operations Manage.

3.2 Розробка структурної схеми

Без мережі неможливо представити роботу будь-якої компанії. Залежність бізнесу від IT технологій буде рости, тому моніторинг і оптимізація продуктивності мережі (Network Performance Monitoring and Diagnostics) дуже важливе завдання, яке стоїть перед IT відділом. Якщо раніше компанії контролювали стан мережевих пристроїв і ключових інтерфейсів за допомогою SNMP і RMON протоколів, то сьогодні важливо розуміти роботу IT-інфраструктури в комплексі і як окремі елементи впливають на продуктивність у локальній мережі. У цей час усе більше запитів з боку корпоративних клієнтів я одержу по контролю над роботою орендованих каналів у різних операторів зв'язку й виконанням операторами угод про якість сервісу (Service Level Agreements).

Моніторинг продуктивності мережі (Network Performance Monitoring) – це програмні й/або апаратні компоненти, що підключаються до мережі для контролю над продуктивністю мережевих компонентів, каналів зв'язку й переданим трафіком.

Застосунки NPM у базовому виконанні поєднують функції збору й зберігання пакетів, переданих по мережі, шляхом копіювання реального трафіку в систему аналізу по основним бізнес застосунків і сервісам. Які застосунки контролювати вирішує кожний замовник для себе сам і вибір здійснюється по комбінації типу транспортного протоколу й номера порту або діапазону портів застосунку. Системи даного класу відрізняються від класичних застосунки Network Monitoring (Ping, SNMP pooling, Syslog і т.д.) тим, що працюють с реальним мережевим трафіком. Інформацію можна аналізувати в реальному часі

з точністю до секунд або аналізувати агреговану статистику за будь-який період часу аж до року.

Застосунки NPM відповідають на запитання: «Як мережа доставляє пакети?», і надають наступні можливості:

- Контроль стану й доступності пристроїв.
- Коректність налаштуванання мережевого устаткування.
- Моніторинг використання каналів зв’язку.
- Аналіз поведінки й роботи застосунків, користувачів.
- Можливість захвата реального мережевого трафіку.
- Оцінка якості голосового й відео трафіку.
- Автоматичні повідомлення про події й перевищення базових параметрів мережі.
- Побудова звітів у ручному або автоматичному режимі.
- Кореляція подій у момент зниження продуктивності.

Ці можливості становлять структурні блоки розроблювальної системи.

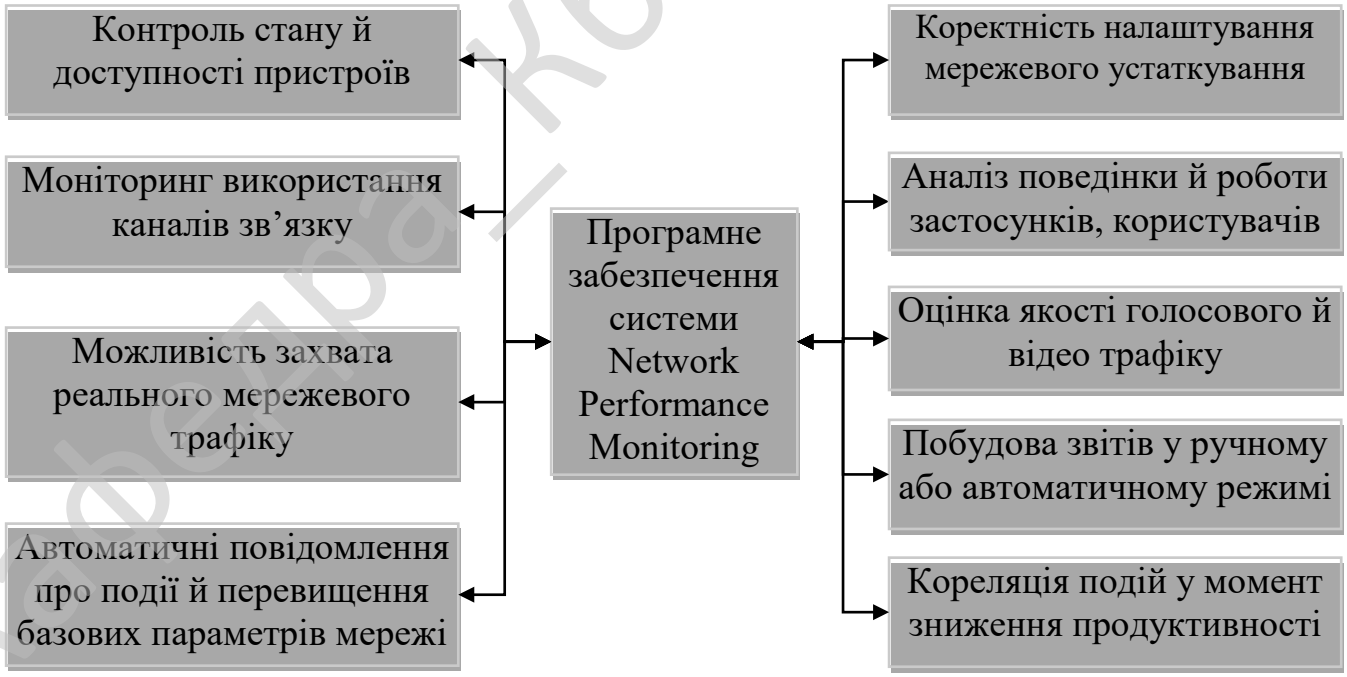

Рисунок 3.1 – Структурна схема системи

Контроль доступності й стану пристроїв виконується за допомогою команди Ping по IP адресі пристрою й/або порту застосунку й за допомогою запитів до MIB базам мережевого устаткування. Результатом виконання даних запитів є статистика у вигляді наочних таблиць або графіків з відповідної колірним кодуванням – зелений (усі добре), жовтий (слід звернути увагу) і червоний (є реальна проблема). Система звертається до стандартних або налаштованим ідентифікаторам об'єктів в MIB базі для одержання необхідної інформації. Також вона може сортувати дані по географії, локальному або вилученому офісу, типу сервісу або застосунку. Найбільш зручні NPM системи зберігають інформацію в багатомірних базах даних, що дозволяє зручно робити докладну деталізацію або налаштовувати фільтри під розв'язуване завдання. Наприклад, тип сервісу, офіси, канали зв'язку, самі повільні застосунки і т.д.

Коректність налаштуванань мережевого устаткування з погляду передачі даних здійснюється на основі аналізу заголовків пакетів і надання статистики по типах застосунків і класів сервісу (CoS), з яким вони передаються й борються за пропускну здатність каналу. Ирафік з мережі клієнта передається із класом сервісу AF2-business, а з мережі приходять пакети із класами AF3-critical, Ef-realtime і AF4-priority, а також наочно надана інформація яку смугу пропущення займає трафік з тим або іншим класом сервісу.

Якщо система NPM і активне устаткування в мережі підтримує Flow технології (NetFlow, IPFIX, sFlow, jFlow і т.д.), то ми можемо контролювати використання каналів зв’язку, проводити аналіз поведінки застосунків і користувачів найпростішим способом. Flow технології вже закладені в куплене активне устаткування, а система NPM може прийняти їх як Flow колектор і надати необхідну статистику.

Flow технології суттєво не навантажують канали й дозволяють у єдиній точці мати докладну інформацію з використання каналів усієї територіально розподіленої мережі й відповісти на такі питання як:

- Які користувачі працюють у мережі і яка тривалість їх сеансів зв’язку?

- Які програми й застосунку вони використовують?
- Хто і які застосунки найбільше використовують канали зв'язку?
- Чи правильно налаштовано моя мережу з точки зору класів сервісів?
- Чи слідують користувачі політиці використання мережі?
- Чи є віруси в моїй мережі, де вони, як вони туди потрапили, куди вони поширюються?

Для передачі голосу по IP використовується транспортний протокол UDP, який не має механізму установки гарантованого з'єднання й підтвердження отриманих пакетів. Це означає, що при роботі даної технології можливі втрати пакетів, затримки в каналах зв'язку, а в сукупності з перевантаженими каналами зв'язку буде однозначно приводити до зниження якості голосу або обриву викликів. Поліпшення якості VoIP вирішується шляхом присвоєння даному сервісу найкращого класу сервісу, тобто найвищого пріоритету, але це веде до симетричного зниження продуктивності інших бізнес-застосунків і сервісів. Отже, система оцінки продуктивності мережі повинна мати можливість оцінювати цей вплив шляхом аналізу статистики по завантаженню інтерфейсів, аналізу CoS по всій корпоративній мережі й на стиках з орендованими каналами зв'язку, а також вести аналіз статистики по якості мови в ідеалі з можливістю відтворення дзвінків

Із чим доводиться зустрічатися зараз при спілкуванні із шановними клієнтами?

Установлена велика кількість розрізнених систем контролю над роботою мережевих пристроїв (Network Monitoring) без прив'язки до ключових сервісів і застосунків. Крім цього спостерігається відсутність якої-небудь кореляції подій, які ведуть до повільної роботи сервісів. Дійсна система моніторингу продуктивності мережі повинна швидко надати відповідь через що гальмує сервіс – мережа, застосунок, сервер і при відсутності IT-фахівців на місці в автоматичному режимі видати повідомлення, опитати збійні елементи мережі по SNMP і захопити трафік, якщо це необхідно.

Таким чином, коли фахівці переступлять до застосунки кейса, у них буде вся достовірна інформація, що відбувалося в мережі, коли користувачі почали скаржитися на повільну роботу сервісу.

Ідеальна система моніторингу за продуктивність мережі повинна:

– Бути масштабована для покриття всієї мережі й будь-яких каналів зв'язку для надання можливості моніторингу за принципом end-to-end, тобто від будь-якого користувача до будь-якого сервісу або застосунку

– Бути всеосяжної й мати можливість нарощування свого функціонала за допомогою ліцензій або додаткових програмних або апаратних агентів.

– Забезпечувати контроль усіх рівнів мережевої моделі OSI і можливість збільшення деталізації аж до автоматичного захвата трафіку, який приводить до перевищення встановлених граничних значень.

У рамках даного розділу я описав сторони, що найбільше часто зустрічаються вимоги з, замовників до систем даного класу. Вибір системи моніторингу продуктивності мережі не тривіальне завдання, і вимагає уважного підходу. Настійно рекомендується проводити пілотні проекти, щоб переконається, що система забезпечує всі заявлені функції й не покладатися на маркетингові описи й гарні слова менеджерів по продажах.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.2. Рішення задачі забезпечення своєчасності передачі інформації в системі Network Performance Monitoring складається з чотирьох етапів:

– визначення множини $\aleph_{баз}$ маршрутів передачі інформації;

– знаходження оптимальної множини маршрутів передачі цифрової інформації в телекомунікаційній мережі $\aleph_{вб}$;

– обчислення коефіцієнтів $\widetilde{k}_s$ розподілу інформаційного потоку і управління навантаженням системи Network Performance Monitoring;

– створення та оновлення таблиці маршрутизації.

При рішенні задачі визначення множини $\aleph_{баз}$ шляхів передачі інформації в системі Network Performance Monitoring для ВЗ «*i*» та «*j*» з множини $\Re$ вузлів зв'язку спочатку необхідно знайти найкоротшу «відстань» (мінімальний час передачі інформаційних пакетів) $T_{i,j\,min}$ від джерела «*i*» до адресата «*j*» і множини $S_j^{(i)}$ вузлів, найближчих ВЗ «*i*» за напрямом руху потоку до «*j*» (множина «вузлів-наступників») у порядку рівнів ієрархії дерева допустимих маршрутів множини U.

При рішенні поставленої задачі відомими алгоритмами пошуку найкоротших шляхів в більшості практичних випадків маємо проблему «зациклення» при передачі інформації в знайдених шляхах. Це призводить до збільшення часу передачі інформаційних пакетів, а деколи і до їх втрати. Уникнути «зациклення» при передачі інформації пропонується шляхом додання обмежень (умова постійної відсутності циклів), які надані у вигляді виразів:

$$T_{k,j} \le T_{i,j\,min}; \tag{3.1}$$

$$T_{k,j\,min} \le T_{i,k,j},\ k \subset R, \tag{3.2}$$

де $T_{k,j\,min}$ – найкоротша «відстань» (мінімальний час передачі інформаційних пакетів) від вузла «*k*» до адресата «*j*»; $T_{i,k,j}$ – «відстань» (час передачі інформаційних пакетів) від вузла «*i*» до адресата «*j*» через вузол «*k*».

Рисунок 3.2 – Функціональна схема системи

Рішення задачі пошуку множини шляхів, що виключають «цикли», складається з двох етапів: визн ачення найкоротшої «відстані» $T_{min\ j,i}$

S_j) «вузлів-наступників» на маршрутах, що

виключають «циклічність», для довільних джерел «*i*» та адресатів «*j*» за порядком множини U рівнів ієрархії дерева вибору допустимих маршрутів.

На першому етапі для визначення найкоротшої «відстані» T_{min} доцільно використати алгоритм розрахунку попередньої топології, який забезпечує вузли зв'язку інформацією про стан зв'язків для обчислення найкоротших шляхів до адресатів. Алгоритм створено на основі відомих алгоритмів стану зв'язків. На відміну від відомих, в цьому алгоритмі враховується ієрархічність побудови системи Network Performance Monitoring (визначаються «відстані» від «вузла-джерела» *i* до «адресата» *j* відповідно до існуючих рівнів ієрархії), що надалі дозволить здійснити пропорційний (з урахуванням коефіцієнтів k_s і k_s) розподіл інформації по знайдених маршрутах.

На другому етапі для знаходження множини шляхів передачі інформації, що виключають «циклічність», використовується алгоритм розповсюдження попередньої топології множини шляхів. На відміну від відомих алгоритмів, в яких не враховується поточний стан ВЗ «*i*» обов'язкова синхронізація обміну службовими повідомленнями про стан зв'язків по всій мережі, в алгоритмі розповсюдження попередньої топології множини шляхів такій синхронізації підлягає тільки один перехід між сусідніми вузлами. Це значно спрощує роботу вузлів зв'язку і скорочує час збіжності алгоритмів. Сукупність алгоритмів розрахунку попередньої топології і розповсюдження попередньої топології множини шляхів є основою способу визначення множини шляхів передачі інформації.

Запропонований спосіб за відсутністю флуктуацій трафіку зіставлений з аналогічними (крива 1 – спосіб Галлагера, крива 2 – модифікований дистанційно-векторний (MDVA)) за часом затримки передачі інформації, проте у декілька разів перевершує їх при різких флуктуаціях вхідного трафіку.

Безпосереднє використання всієї знайденої множини $\aleph_{баз}$

$N_{баз}$ шляхів деякої (оптимальної) сукупності $N_{вб}$ маршрутів, використання якої в умовах обмежень, що накладаються, дозволить забезпечити максимально можливу достовірність передачі інформації.

В умовах, описаних вище, одержані криві залежності ймовірності ($_{баз}, \Delta t$) від λ. Збільшення числа M використаних маршрутів, призводить до істотного (у 2,3…3,4 разів) збільшення ймовірності ($_{баз}, \Delta t$) спотворення інформації в процесі її передачі. Слід особливо відзначити, що для відомих методів розподілу характерна відсутність моніторингу поточного завантаження маршрутів, змін вхідного потоку інформації, а інколи і технічного стану каналів зв'язку. Принцип рівномірного завантаження вибраних M маршрутів, який використовується в таких методах, призводить до перевантаження одних і до неефективного використання інших маршрутів.

Для визначення початкового завантаження підмережі системи Network Performance Monitoring пропонується проводити розрахунок коефіцієнтів k_s ^{j,} розподілу потоку інформації від джерела «i» до адресата «j» між «вузлами-наступниками» «k» з M-мірної множини S_j $^{)}$ залежно від значень «відстані» (часу передачі пакетів) між ВЗ на маршруті за співвідношенням

$$k_s \;^{,j,\,ki} \left(1 \;\Big(\quad + \;_{,ki} \Big) \Big/ - \sum_{\xi \in S_j^{i)}(} \Big(= \; + \;_{i,\xi} \Big) \right)_j \Big/ \;_{,} \Big(\xi \Big| S_j^{i)} \Big| (\;\; 1 \Big) \, t \quad \in T S_j^{)i}, \qquad (3.3)$$

де $T_{,j}$ – час передачі пакетів від «вузла-наступника» «k» до адресата «j» на маршруті (i, j); $t_{\,ki}$ – «відстань» від джерела «i» до «вузла-наступника» «k»; $\left|S_j^{\,i)}\right|$ – потужність множини $S_j^{\,i)}$.

Розраховані залежності середнього часу $T_{срд}$

$Q^{сч)}$ їхньої доставки за час, що не перевищує допустиме значення, від інтенсивності λ вхідного потоку інформації для повнозв'язного фрагменту системи Network Performance Monitoring в умовах мінімальної $(\rho_{min} = 16$ Кбіт/с$)$, максимальної $(\rho_{\max} = 30$ Кбіт/с$)$, середньої $(\rho_z = 19$ Кбіт/с$)$ пропускної спроможності каналів зв'язку (умови (I) – суцільні криві), $\rho_{min} = 14$ Кбіт/с, $\rho_{\max} = 300$ Кбіт/с, $\rho_z = 70$ Кбіт/с (умови (II) – пунктирні криві) при кількості маршрутів M=5. Параметром сімейства кривих є метод розподілу мережевого ресурсу в системі Network Performance Monitoring («1» – статичний метод, «2» і «3» – відповідно відомий (MDVA) і адаптивний до умов початкового завантаження мережі методи розподілу).

Аналіз залежностей показав, що у деяких ситуаціях адаптивний розподіл потоку інформації дозволяє «заощадити» 2 і більш маршрутів і тим самим додатково понизити ймовірність ($_{баз}, \Delta t$) порівняно з відомими методами управління, що свідчить про необхідність оптимізації топології підмережі системи Network Performance Monitoring. В роботі пропонується метод знаходження оптимальної множини маршрутів в системі Network Performance Monitoring, який включає постановку завдання та процедуру знаходження оптимальної множини маршрутів в системі Network Performance Monitoring. При цьому визначення початкового завантаження підмережі дозволяє найкращим чином розподілити потік пакетів, тим самим мінімізуючи $T_{сред}$ і максимізуючи $Q^{сч)}$ в даних умовах.

В подальшому, для передачі інформації про повітряну обстановку та забезпечення «збалансування» загрузки при флуктуаціях трафіку виконується процедура розподілу інформаційних пакетів за оптимальною множиною маршрутів в системі Network Performance Monitoring.

Проведемо оцінки ефективності методу забезпечення своєчасності передачі інформації в системі Network Performance Monitoring при передачі інформації про повітряну обстановку і вірогідності отриманих результатів. В якості рекомендацій щодо практичного застосування методу забезпечення своєчасності передачі

інформації в системі Network Performance Monitoring при управлінні повітряним рухом розроблена структура процесу передачі інформації про повітряну обстановку в режимі реального часу. Врахована необхідність здійснювати комплекс заходів (адаптивне кодування, зменшення статистичної і психовізуальної надмірності, адаптивна маршрутизація та ін.), направлених на управління швидкістю передачі інформації про повітряну обстановку і зниження її інтенсивності. Для компенсації втрачених при передачі в каналах зв'язку інформаційних пакетів і забезпечення безперервності відтворення інформації про повітряну обстановку розроблено алгоритм компенсації втрачених інформаційних пакетів. Це дозволить використовувати для управління передачею цифрової інформації про повітряну обстановку протоколи транспортного рівня (RTP і UDP).

Оцінка ефективності роботи методу забезпечення своєчасності передачі інформації в системі Network Performance Monitoring по відношенню до відомих методів показала ряд його переваг (ймовірність доведення інформації до одержувача за час, що не перевищує допустиме значення, вище за аналогічну ймовірність відомих методів до 3 разів, середній час доставки інформаційних пакетів менше аналогічної характеристики відомих методів до 15 разів).

Для обґрунтування достовірності отриманих результатів математичного моделювання проведено імітаційне моделювання передачі інформації в повнозв'язній комп'ютерній мережі в умовах застосування бездротових каналів зв'язку (середня пропускна спроможність $\rho_z = 19$Кбіт/с) при різних інтенсивностях вхідного потоку (у діапазоні $\lambda = [5,...,30]$ Кбіт/с).

За критерієм згоди Персона з рівнем значущості $\alpha = 0{,}01$ показано, що число прийнятих за час $T_{доп}$ інформаційних пакетів а також ймовірність $Q_{експ}^{(сч)}$ можна вважати розподіленими за нормальним законом.

Одержані довірчі інтервали для оцінок математичного очікування $Q_{експ}^{(сч)}$, в які потрапляють «розрахункові» значення $Q_{\lambda}^{(сч)}$ з довірчою ймовірністю 0,95.

Високий ступінь збігу результатів імітаційного і математичного моделювання підтверджує достовірність математичної моделі, використаної для розрахунку ймовірності $Q^{(сч)}$ доставки інформаційних пакетів за час, що не перевищує допустиме значення.

3.4 Розробка діаграми процесів

Відповідно до методичних рекомендацій розроблення графічної частини кваліфікаційної бакалаврської роботи розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3.

Розроблена діаграма взаємодії процесів використовується для представлення та візуалізації процесів обробки даних тобто структурного проектування бакалаврської роботи.

Основні складові елементи діаграми взаємодії процесів це потоки даних:

– Репозиторії, потік сховища даних.

– Потоки зовнішні по відношенню до системи сутності.

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Потоки даних гібридні між елементами трьох попередніх типів.

Відповідно до документації основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей.

Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників. Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

Рисунок 3.3 – Діаграма взаємодії процесів

Для схематичного представлення системи що розробляється необхідно спочатку представити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи в цілому у подальшому. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Розроблена діаграма взаємодії процесів системи в подальшому уточнюється шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Таким чином у результаті після розгляду, вищеописаної системи, схеми структурної, функціональної, діаграми взаємодії процесів перейдемо до опису та розгляду блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.1. З рисунку видно, що після запуску програми спочатку відбуваються наступні дії:

– Виведення вікна моніторингу NPM.

– Ініціалізація структури поточного звіту з використанням часових значень.

– Підпрограма сканування мережі.

– Підпрограма заповнення звітності.

– Підпрограма оновлення записів та графіків.

– Підпрограма оновлення поточних значень параметрів.

– Підпрограма формування таблиць маршрутизації.

– Підпрограма запису звітності у файл.

– Підпрограма фрагментації інформаційного пакету.

– Підпрограма оновлення таблиць маршрутизації.

– Приховання головного вікна вікна моніторингу NPM.

На рисунку 4.2 зображена робота підпрограми у якої відбуваються наступні дії:

– Сканування першої хвилі трафіку комп'ютерної мережі.

– Заповнення структур даних трафіком вхідних пакетів даних.

– Оцінка вхідних пакетів даних, знаходження множин поточних маршрутів.

– Друге сканування та, обробка трафіку вхідних пакетів.

– Формування звіту поводження трафіку вхідних пакетів даних.

– Створення та формування образу характеру розвитку трафіку вхідних пакетів даних.

– Визначення частотних показників ЛМ, формування даних для графіку.

– Формування поточного звіту сканування трафіку.

– Виведення поточного результату сканування.

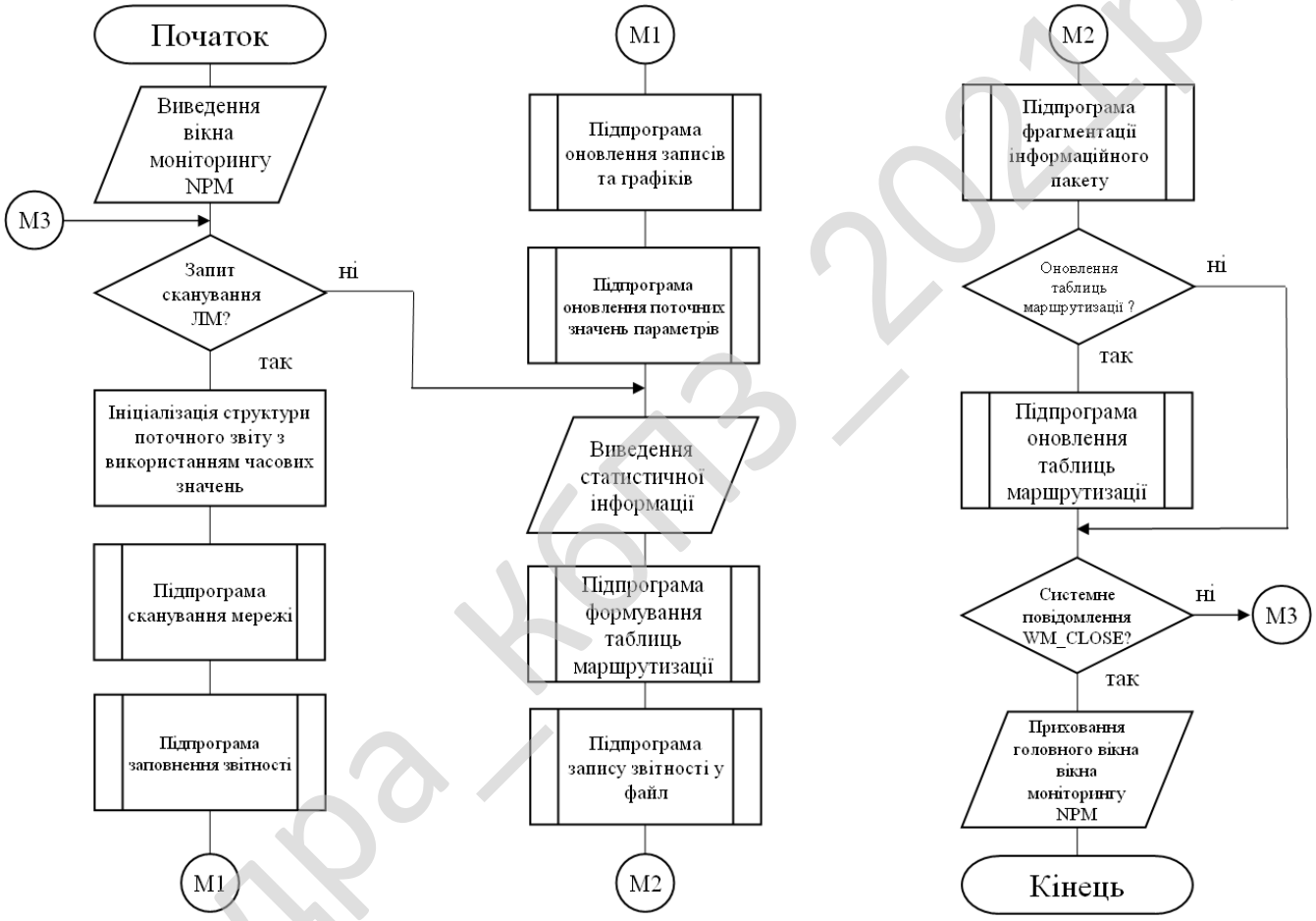

Рисунок 4.1 – Блок-схема основної програми

Розглянемо вихідний код який необхідний для посилання ECHO-запиту для визначення поточного статусу ПК:

```
procedure TForm19.Button1Click(Sender: TObject);
var // змінні процедури системи
    hIP20 : THandle;
    pingBuffer : array [0..31] Char;
```

```
    pIpe : ^icmp_echo_reply;
    pHostEn : PHostEnt;
    wRequested : WORD;
    lwsaData : WSAData;
    ER : DWORD;
    destAddress : In_Addr;
begin
// Початок процедури, дії по роботі з handle
    hIP20 := IcmpCreateFile();
GetMem( pIpe, sizeof(icmp_echo_reply)+ sizeof(pingBuffer));
    pIpe.Data := @pingBuffer;
    pIpe.DataSize := sizeof(pingBuffer);
    wRequested := MakeWord(1,1);
    ER := WSAStartup(wRequested,lwsaData);
    if (ER <> 0) then
    begin
Memo1.SetTextBuf('ER in call to '+'WSAStartup().');
Memo1.Lines.Add('ER code: '+IntToStr(ER));
    Exit;
    end;
        pHostEn := gethostbyname;
    ER := GetLastER();
    if (ER <> 0) then
    begin
// результат у поле Memo1
Memo1.SetTextBuf('ER in call to'+'gethostbyname().');
Memo1.Lines.Add('ER code: '+IntToStr(ER));
    Exit;
    end;
destAddress := PInAddr(pHostEn^.h_addr_list^)^;
// Посилаємо ping-пакет
Memo1.Lines.Add('Pinging'+pHostEn^.h_name+' ['+
    inet_ntoa(destAddress)+'] '+ ' with '+
    IntToStr(sizeof(pingBuffer))+
'bytes data:');
IcmpSendEcho(hIP20destAddress.S_addr, pingBuffer sizeof(pingBuffer), Nil, pIpe
sizeof(icmp_echo_reply)+ sizeof(pingBuffer), 5000);
ER := GetLastER();
    if (ER <> 0) then
    begin
Memo1.SetTextBuf('ER in call to '+'IcmpSendEcho()');
Memo1.Lines.Add('ER code: '+IntToStr(ER));
```

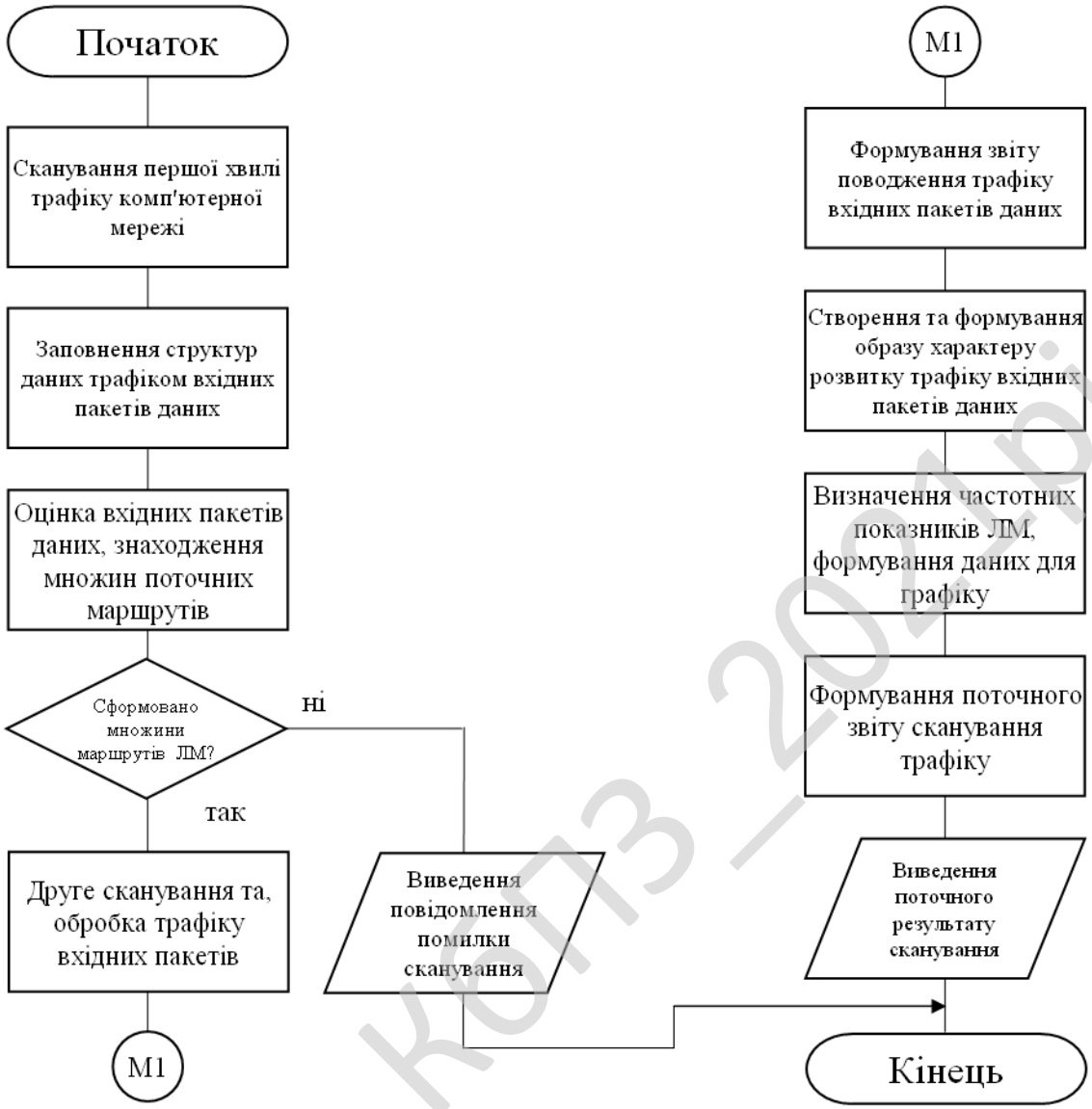

Рисунок 4.2 – Блок-схема роботи підпрограми

```
   Exit;
    end;
// Дивимося деякі з даних, що повернулися
Memo1.Lines.Add('Reply from '+ IntToStr(LoByte(LoWord(pIpe^.Address)))+'.'+
IntToStr(HiByte(LoWord(pIpe^.Address)))+'.'+
IntToStr(LoByte(HiWord(pIpe^.Address)))+'.'+
IntToStr(HiByte(HiWord(pIpe^.Address))));
Memo1.Lines.Add('Reply time:'+IntToStr(pIpe.RTTime)+' ms');
    IcmpCloseHandle(hIP20);
    WSACleanup();
    FreeMem(pIpe);
end;
```

Реалізація програмного опитування користувачів. При створенні ПЗ в мережі для своєчасної передачі інформації в комп'ютерних мережах в деяких випадках необхідно знати поточний стан як локального, так і видалених хостів (чи має локальний хост в даний момент можливість виходу в мережу Інтеренет, чи доступний якийсь видалений хост і т.д.).

Загальновідомо, що для вказаної мети використовується утиліта ping. Принцип роботи ping-а заснований на використовуванні протоколу ICMP – Internet Control Message Protocol (протокол керівників, або контрольних, повідомлень). Одним з типів повідомлень протоколу є "відлуння-запиту чи як ще кажуть луна-запит". Отримавши "луна-запит" хост зобов'язаний відповісти тому, що послав "луна-відповіддю".

Розглянемо існуючий код визначення типів і прототипи функцій для Microsoft icmp.dll (PING):

```
type
// Інформація заголовка IP (Наповнення)
ip_option_information = packed record
        Ttl : byte;
// Час життя (використовується traceroute-ом)
        Tos : byte;
// Тип обслуговування, звичайно 0
        Flags : byte;
// Прапори заголовка IP, звичайно 0
// Розмір даних в заголовку, звичайно 0, максимум 40
        OptionsSize : byte;
        OptionsData : Pointer;
// Покажчик на дані
end;
icmp_echo_reply = packed record
        Address : u_long;
// Адреса відповідаючого
        Status : u_long;
// IP_STATUS
        RTTime : u_long;
// в мілісекундах
        DataSize : u_short;
        Reserved : u_short;
```

```
// Зарезервовано
        Data : Pointer;
// Покажчик на повернені дані
// Інформація із заголовка IP
        Options : ip_option_information;
    end;
    PIPINFO = ^ip_option_information;
    PVOID = Pointer;
function IcmpCreateFile():THandle;stdcall;external 'ICMP.DLL';
function IcmpCloseHandle(IcmpHandle : THandle): BOOL; stdcall; external 'ICMP.DLL'
name 'IcmpCloseHandle';
function IcmpSendEcho(IcmpHandle : THandle;
// Адреса одержувача (в мережному порядку)
DestAddress : u_long;
RequestData : PVOID;
// Покажчик на посилані дані
RequestSize : Word;
// Розмір даних що посилається
RequestOptns : PIPINFO;
// Покажчик на посилану структуру
// ip_option_information (може бути nil)
// Покажчик на буфер, що містить відповіді.
ReplyBuffer:PVOID;
ReplySize : DWORD;
//Розмір буфера відповідей
Timeout : DWORD
) : DWORD; stdcall; external 'ICMP.DLL' name 'IcmpSendEcho';
// Час очікування відповіді в мілісекундах
```

Наведемо код классу TNetwork. Це клас, який створює список всіх комп'ютерів в робочій групі. Цей клас є нащадком класу TStrings і повністю сумісний з іншим нащадками цього класу.

Об'єкти цього класу записуються у властивості об'єктів класу TNetworkNeiborhood. Клас TNetworkNeiborhood містить об' єкти і властивості робочих груп.

```
TNetwork = class (TStringObjectList);
  TNetworkNeighborhood = class (TStringObjectList)
// TNetworkNeiborhood – клас, який створює список всіх робочих груп в мережі
  private
    function CreatePIDL(Size: Integer): PItemIDList;
```

```
    procedure DisposePIDL(ID: PItemIDList);
    function NextPIDL(IDList: PItemIDList): PItemIDList;
    function GetPIDLSize(IDList: PItemIDList): Integer;
    function CopyPIDL(IDList: PItemIDList): PItemIDList;
    procedure StripLastID(IDList: PItemIDList);
    function GetPrevPIDL(PIDL: PItemIDList): PItemIDList;
    class function GetDisplayName(ShellFolder: IShellFolder; PIDL: PItemIDList):
TString;
    function OriginFolder:  IShellFolder;
    function OriginFolderNT: IShellFolder;
    class function EnumObjects(ShellFolder: IShellFolder): IEnumIDList;
    class procedure ParseFolder(Folder: IShellFolder; Items: TStringObjectList;
StorePIDLs: Boolean = False);
    class procedure ParseFolderEx(Folder: IShellFolder; Items: TStrings);
    function FreeRefObj(Index: Integer; var Obj: TStringObject): Integer;
    function GetWorkgroup(Name: TString): TNetwork;
  public
{ Процедура Оновлення шукає всі доступні робочі групи в мережі }
    procedure Refresh;
{ містить списки всіх комп'ютерів в  мережі}
    property Workgroup[Name: TString]: TNetwork read GetWorkgroup;
{ Функція FindComputer шукає комп'ютер зі вказаним ім'ям і повертає ім'я робочої
групи де його знайдено, або порожню строку
якщо комп'ютера з таким іменем у мережі немає }

function FindComputer(Name: TString): TString;
{ Процедура ListComputers копіює список всіх комп'ютерів в мережі
у об'єкті TStrings}
procedure ListComputers(Strings: TStrings);
{ Процедура ListNetwork копіює відсортований в алфавітному порядку список всіх
робочих груп і комп'ютерів в мережі.
Робочі групи мають ' TObject(1)' у відповідному елементі властивості
об'єктів, а комп'ютери – ' nil'}
    procedure ListNetwork(Strings: TStrings);
    function Add(const S: string): Integer; override;
    procedure Clear; override;
    procedure Delete(Index: Integer); override;
    procedure Insert(Index: Integer; const S: string); override;
    constructor Create;
  end;
```

Розглянемо використану архітектуру клієнт-сервер. Це є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних програм і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

– набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;

– набір клієнтів, які використовують сервіси, що надаються серверами;

– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери – це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми – і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

– рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;

– прикладний рівень, який реалізує основну логіку ПЗ і на якому здійснюється необхідна обробка інформації;

– рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

– модель тонкого клієнта, в рамках якої вся логіка ПЗ та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;

– модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Типовим прикладом клієнт-серверної взаємодії є WWW. Існує величезна кількість веб-серверів, на яких розміщується та чи інша інформація. У найпростішому випадку ця інформація являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML. Але ситуація, як правило, є складнішою; значна частина веб-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

Для того, щоб людина, яка працює в Інтернеті, могла переглянути ту чи іншу сторінку, на її комп'ютері повинно бути встановлено відповідне програмне забезпечення. Програми для перегляду веб-сторінок називаються браузерами.

Але, крім браузерів, до серверів можуть звертатися і інші клієнти, а саме – автономні програми. Вони можуть передбачати взаємодію з людиною, а можуть працювати в цілком автоматичному режимі. Типовим класом таких програм є роботи, призначені для автоматичного перегляду веб-ресурсів. Зокрема, роботи є важливим елементом пошукових систем і використовуються ними для перегляду сторінок і збору інформації про них.

Для запиту до веб-сервера клієнтська програма повинна задати місцезнаходження комп'ютера, на якому розміщується серверна програма, назву потрібного документа і, можливо, інші дані, які специфікують запит. Мережа забезпечує знаходження сервера і передачу йому клієнтського запиту. Серверні програми обробляють цей запит, відповідь пересилається по мережі клієнтові.

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка ПЗ. Програми проміжного рівня можуть функціонувати під управлінням спеціальних серверів ПЗ, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

Для роботи з системою користувач використовує стандартне програмне забезпечення –звичайний браузер. Це позбавляє його необхідності завантажувати та інсталювати спеціальні програми (хоча інколи така необхідність все-таки виникає).

Але користувачеві слід надати в розпорядженні інтерфейс, який дозволяв би йому взаємодіяти з системою і формувати запити до неї. Форми, що визначають цей інтерфейс, розміщуються на веб-сторінках та завантажуються разом з ними.

Веб-оглядач формує запит та пересилає його до сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних.

Сервер даних здійснює операції з даними, що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних.

Найчастіше веб-сервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоч і являють собою окремі і логічно незалежні програмні модулі.

На сучасному етапі для програмування модулів проміжного рівня використовується мова серверних сценаріїв PHP, а для управління даними – СУБД MySQL. Таким чином, зв'язку PHP-MySQL слід розглядати як стандартний інструмент для створення порівняно простих інтерактивних веб-сайтів та систем електронної комерції; близько 90% комерційних систем сьогодні створюється саме на цій основі. Водночас як засоби управління даними, так і middleware-засоби можуть бути найрізноманітнішими. Так, для створення серверних програм, крім PHP, широко застосовуються Java, Perl, Python, Delphi.

Взагалі, технології створення розподілених, зокрема веб-програм, стрімко розвиваються. Слід згадати про технології EJB (Enterprise Java Beans), CORBA, а також про .NET – порівняно нову ініціативу компанії Microsoft. Для зберігання даних та їх передачі часто використовується так звана розширювана мова розмітки XML (Extensible Markup Language).

4.2 Захист розробленого програмного забезпечення

Дані в програмі захищаються за допомогою використання алгоритму шифрування ДСТ Р 34.10-2012, який використовує перетворення у групі точок на еліптичній кривій в простому полі Галуа. У системі шифрування/дешифрування як параметри розглядається еліптична крива E_p (a,b) і точка G на ній. Учасник В вибирає закритий ключ n і обчислює відкритий ключ $P_B = n \times G$. Щоб

зашифрувати повідомлення P_m використовується відкритий ключ одержувача В P_B. Учасник А вибирає випадкове ціле позитивне число k і обчислює зашифроване повідомлення C_m, що є точкою на еліптичній кривій.

$$C_m = \{k \times G, P_m + k \times P_B\} \quad (4.1)$$

Щоб дешифрувати повідомлення, учасник В множить першу координату точки на свій закритий ключ і віднімає результат від другої координати:

$$P_m + k \times P_B - n_B \times (k \times G) = P_m + k \times (n_B \times G) - n_B \times (k \times G) = P_m \quad (4.2)$$

Учасник А зашифрував повідомлення P_m додаванням до нього $k \times P_B$. Ніхто не знає значення k, тому, хоча P_B і є відкритим ключем, ніхто не знає $k \times P_B$. Супротивнику для відновлення повідомлення доведеться обчислити k. Зробити це буде нелегко. Одержувач також не знає k, але йому як підказку посилається $k \times G$. Помноживши $k \times G$ на свій закритий ключ, одержувач одержить значення, що було додано відправником до незашифрованого повідомлення. Тим самим одержувач, не знаючи k, але маючи свій закритий ключ, може відновити незашифроване повідомлення.

$p>3$ є непарним числом. Еліптична крива GF(p) над F_p виражена формулою виду

$$y^2 = x^3 + a \cdot x + b \quad (4.3)$$

де $a, b \in F_p$ та $4 \cdot a^3 + 27 \cdot b^2 \neq (\mathrm{mod}\ p)$

Додавання точок.

$P = (x_1, y_1) \in E(F_p)$ та $Q = (x_2, y_2) \in E(F_p)$, де $P \neq \pm Q$.

Тоді $P + Q = (x_3, y_3)$, де:

$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \text{ та } y_3 = (\frac{y_2 - y_1}{x_2 - x_1}) \cdot (x_1 - x_3) - y_2 \quad (4.4)$$

Подвоєння точки.

Припустимо $P = (x_1, y_1) \in E(F_p)$, де $P \neq -P$. Тоді $2P = (x_3, y_3)$, де

$$x_3 = (\frac{3 \cdot x_1^2 + a}{2 \cdot y_1})^2 - 2 \cdot x_1 \text{ та } y_3 = (\frac{3 \cdot x_1^2 + a}{2 \cdot y_1}) \cdot (x_1 - x_3) - y_1 \quad (4.5)$$

Також використовується хеш функція H() визначена в алгоритмі ГОСТ 28147-89. числа p, q, a, y – є відкритими для усіх користувачів мережі , число x є секретним.

Щоб підписати деяке повідомлення m потрібні такі кроки :

1) Користувач А генерує випадкове число k, $k<q$.

2) Користувач А обчислює значення r, s за формулами :

$$r = (a^{K} \bmod p) \bmod q, \quad (4.6)$$

$$s = (x * r + k * H(m)) \bmod p. \quad (4.7)$$

Якщо $H(m)$ кратне q, то $H(m) \bmod q = 1$.

3) Якщо $r = 0$ то (беремо інше k) перехід до 1).

4) Цифровий підпис являє собою два числа $r \bmod 2^{256}$ та $s \bmod 2^{256}$.

Користувач А відправляє користувачу В повідомлення m із цим цифровим підписом.

Користувач В перевіряє отриманий підпис виконуючи наступні кроки :

1) $$V = H(m)^{q-2} \bmod q; \quad (4.8)$$

$$Z1 = (s * v) \bmod q; \quad (4.9)$$

$$Z2 = ((q - r) * v) \bmod q; \quad (4.10)$$

$$U=((a^{Z1} * y^{Z2}) \bmod p) \bmod p. \quad (4.11)$$

2) Якщо $u = r \, 2^{256}$, то підпис істиний.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1. З рисунку головного вікна можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Функціональних кнопок ПЗ: Оновлення; Налаштування; Параметри сканування; Звіти.

– Верхнього меню: Файл; Налаштування; Параметри мережі; Шаблон запланованих дій; Довідка.

– Розділу обрання групи: Поточної інформації; Підпункту функціональних кнопок; Поточна завантаженість мережі.

– Розділу виведення результату роботи системи у вигляді вікон реалізації функцвоналу.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші, де відображаються підпункти журнал роботи та довідка.

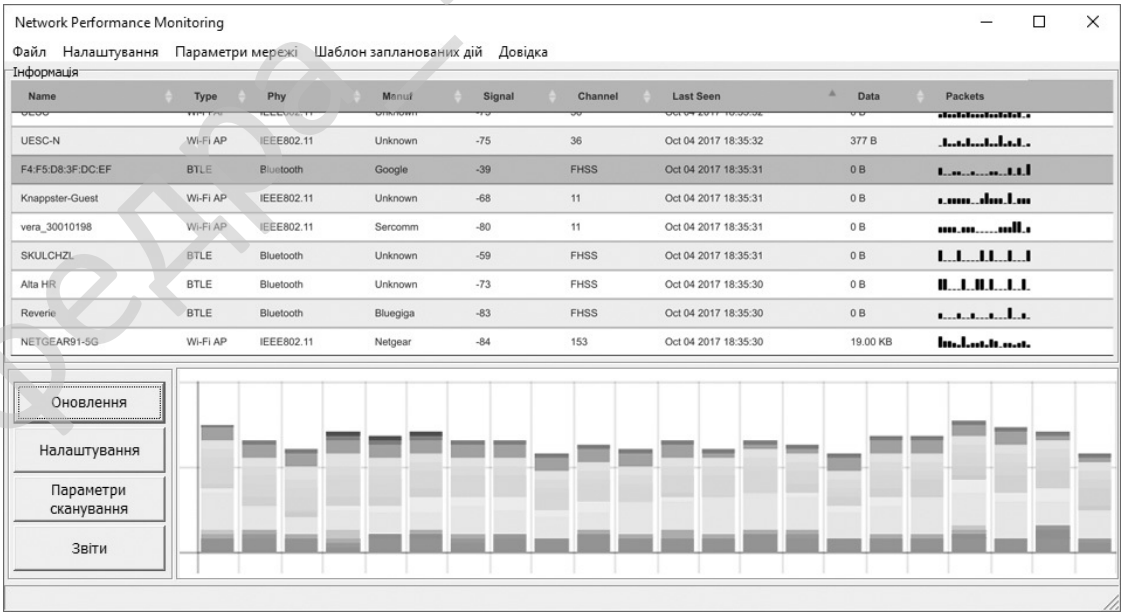

Рисунок 5.1 – Головне вікно ПЗ

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення. Розроблена програма має дуже простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий. Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

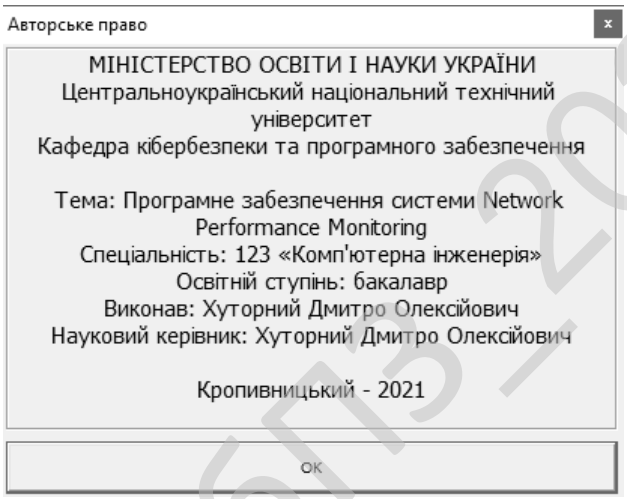

Рисунок 5.2 – Авторське право

Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Таким чином у результаті вищерозглянутого можна стверджувати що розроблено інтерфейс системи у відповідності з вибраною метою роботи. Система містить максимальний необхідний набір функцій придатних для виконання будь-яких дій для забезпечення повноцінної роботи програми. Далі розглянемо висновки та використані літературні джерела.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної бакалаврської роботи, призначено для системи Network Performance Monitoring.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем Network Performance Monitoring.

– Досліджена система Network Performance Monitoring.

– На основі отриманих результатів досліджень створена програмна реалізація системи Network Performance Monitoring.

Розроблені під час виконання кваліфікаційної бакалаврської роботи алгоритми дозволяють успішно вирішувати завдання Network Performance Monitoring.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Delphi 10.4 Sydney. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи Network Performance Monitoring. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ДСТ Р 34.10-2012.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Конахович Г.Ф. Сети передачи пакетных данных / Г.Ф. Конахович, В.М.Чуприн. – К.:МК-Пресс, 2006. – 272 с.

2. Королев А.В. Адаптивная маршрутизация в корпоративных сетях / А.В. Королев, Г.А. Кучук, А.А. Пашнев. – Х.: ХВУ, 2003. – 224 с.

3. Кучерявый Е.А. Управление трафиком и качество обслуживания в сети Интернет / Евгений Андреевич Кучерявый. – СПб.: Наука и техника, 2004. – 336 с.

4. Кучук Г.А. Управление ресурсами инфотелекоммуникаций / Г.А. Кучук, Р.П. Гахов, А.А. Пашнев. – М.: Физматлит, 2006. – 220 с.

5. Лагутин В.С., Степанов С.Н. Телетрафик мультисервисных сетей связи / В.С. Лагутин, С.Н. Степанов. – М.: Радио и связь, 2000. – 320 с.

6. Майника Э. Алгоритмы оптимизации на сетях и графах: пер. с англ. / Э. Майника; под ред. Е.К. Масловского. – М.: Мир, 1981. – 321 с.

7. Мохамад Гани Абу Таам Разработка математической GERT-модели технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / А.А.Смирнов, Мохамад Гани Абу Таам // Информационные системы в управлении, образовании, промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Вид-во ТОВ «Щедра садиба плюс», 2014. – 498 с.

8. Мохамад Гани Абу Таам Метод управления доступом в интеллектуальных узлах коммутации / Мохамад Гани Абу Таам, А.А.Смирнов // Информационные технологии и защита информации в информационно-коммуникационных системах: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Вид-во ТОВ «Щедра садиба плюс», 2015. – 486 с.

9. Мохамад Гани Абу Таам Математическая GERT-модель технологии передачи метаданных в облачные антивирусные системы / В.В.Босько, А.А.Смирнов, И.А.Березюк, Мохамад Гани Абу Таам // Збірник наукових праць "Системи обробки інформації". – Випуск 1(117). – Х.: ХУПС – 2014. – С. 137-141.

10. Мохамад Гани Абу Таам Структурно-логическая GERT-модель технологии распространения компьютерных вирусов / А.А.Смирнов, И.А.Березюк, Мохамад Гани Абу Таам // Системи управління, навігації та зв'язку. – Випуск 1(29). – П.: ПНТУ. – 2014. – С. 120-125.

11. Мохамад Гани Абу Таам Сравнительные исследования математических моделей технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Гани Абу Таам, А.А. Смирнов, А.В. Коваленко, С.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 9(125). – Х.: ХУПС – 2014. – С. 105-110.

12. Мохамад Гани Абу Таам Математическая модель интеллектуального узла коммутации с обслуживанием информационных пакетов различного приоритета / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 4 (41). – Харків: ХУПС. – 2014. – С. 48-52.

13. Мохамад Гани Абу Таам Исследование показателей качества функционирования интеллектуальных узлов коммутации в телекоммуникационных системах и сетях / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 4(17). – Харків: ХУПС. – 2014. – С.90-95.

14. Мохамад Гани Абу Таам Усовершенствованный алгоритм управления доступом к «облачным» телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 1(126). – Х.: ХУПС – 2015. – С. 150-153.

15. Мохамад Гани Абу Таам Анализ и исследование методов управления сетевыми ресурсами для обеспечения антивирусной защиты данных / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Системи озброєння і військова техніка. – Випуск 3(43) – Х.: ХУПС – 2015. – С. 100-107.

16. Мохамад Гани Абу Таам Исследование эффективности метода управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 3(19). – Х.: ХУПС. – 2015. – С. 134-141.

17. Mohamad Abou Taam Method of controlling access to intellectual switching nodes of telecommunication networks and systems / A.A. Smirnov, Mohamad Abou Taam, S.A. Smirnov // International Journal of Computational Engineering Research (IJCER). – Volume 5, Issue 5. – India. Delhi. – 2015. – P. 1-7.

18. Мохамад Гани Абу Таам GERT-модель технологии передачи данных в облачные антивирусные системы / А.А. Смирнов, В.В. Босько, Мохамад Гани Абу Таам // Збірник тез доповідей науково-практичної конференції «Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку». м. Харків. 12-13 березня 2014 р. – Харків. АВВ МВС. – 2014. – С. 18-19.

19. Мохамад Гани Абу Таам Математическое моделирование технологии передачи сигнатур в облачные антивирусные системы / Мохамад Гани Абу Таам, А.А. Смирнов // Збірник тез VI міжнародної науково-практичної конференції “Проблеми і перспективи розвитку ІТ-індустрії”. м. Харків. 17-18 квітня 2014 р. – Харків: ХНЕУ. – 2014. – С. 260.

20. Мохамад Гани Абу Таам Анализ требований к качеству обслуживания в информационно-телекоммуникационных системах / А.А. Смирнов, Мохамад Гани Абу Таам // Збірник тез XVI міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх

застосування». м. Кіровоград. 11-12 квітня 2014 р. – Кіровоград: КНТУ. – 2014. – С. 124-126.

21. Мохамад Гані Абу Таам Дослідження та реалізація GERT-моделі технології розповсюдження комп'ютерних вірусів для захисту телекомунікаційних систем / Мохамад Гані Абу Таам, С.А. Смирнов // Збірник тез науково-практичної конференції «Інформаційні технології та комп'ютерна інженерія». м. Кіровоград. 4 грудня 2014 р. – Кіровоград: КНТУ. – 2014. – С. 168.

22. Мохамад Гани Абу Таам Исследование математических моделей технологии распространения компьютерных вирусов / А.А. Смирнов, Мохамад Гани Абу Таам, С.А. Смирнов // Збірник наукових праць міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 25-28 лютого 2015 р. – Київ: Європейський університет. – 2015. – С. 90-91.

23. Мохамад Гани Абу Таам Метод управления доступом к «облачным» ресурсам для защиты телекоммуникационных систем / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез всеукраїнської науково-практичної конференції «Інформаційна безпека держави, суспільства та особистості». м. Кіровоград. 16 квітня 2015. – Кіровоград: КНТУ. – 2015. – С. 50-52.

24. Мохамад Гани Абу Таам Разработка метода управления доступом в интеллектуальных узлах коммутации / А.А. Смирнов, Мохамад Гани Абу Таам, С.А. Смирнов // Збірник тез VII міжнародної науково-практичної конференції "Проблеми і перспективи розвитку IT-індустрії". м. Харків. 17-18 квітня 2015 р. – Харків: ХНЕУ. – 2015. – С. 14.

25. Мохамад Гани Абу Таам Реализация метода управления доступом в интеллектуальных узлах коммутации / А.А. Смирнов, Мохамад Гани Абу Таам // Збірник тез XVII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 17-18 квітня 2015 р. – Кіровоград: КНТУ. – 2015. – С. 91-92.

26. Мохамад Гани Абу Таам Реализация математической модели интеллектуального узла коммутации для обеспечения защищенности телекоммуникационной сети / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез II Міжнародної науково-практичної Інтернет-конференції «Інформаційна та економічна безпека» (INFECO-2015)». м. Харків. 21-22 травня 2015 р. – Харків: ХІБС УБС НБУ. – 2015. – С. 20-24.

27. Мохамад Гани Абу Таам Разработка математической модели технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Сборник тезисов XI международной конференции "Стратегия качества в промышленности и образовании". г. Варна. Болгария. 01 – 06 июня 2015 г – Варна. ТУВ. – 2015. – С. 488-491

28. Мохамад Гани Абу Таам Метод управления доступом к облачным телекоммуникационным ресурсам для обеспечения защиты данных / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Комп'ютерні технології та інформаційна безпека». м. Кіровоград. 2-3 липня 2015 р. – Кіровоград: КНТУ. – 2015. – С. 4-5.

29. Мохамад Гани Абу Таам Имитационная модель системы управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез першої всеукраїнської науково-практичної конференції «Перспективні напрями захисту інформації». м. Затока. 7-9 вересня 2015 р. – Одеса: ОНАЗ. – 2015. – С. 90-94.

30. МСЭ-Т Рекомендация G.101. Международные телефонные соединения и цепи – Общие определения //11/2003. [Електронний ресурс]. – Режим доступу до ресурсу: http://www. telecom61.ru›SharedFiles/Download.aspx?…pageid=106

31. Одом Ш. Коммутаторы CISCO / Ш. Одом, Х. Ноттингем – М.:"Кудиц-Образ", 2003. – 528 с.

32. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов / В.Г. Олифер, Н.А. Олифер. –2-е изд. – СПб.: Питер, 2007. – 958 с.

33. Руководство по технологиям объединенных сетей. 4-е изд. / пер.с англ. и ред. А.Н. Крикуна – М.: Изд. дом «Вильямс», 2005. – 1040 с.

34. Свами М.Н., Тхуласираман К. Графы, сети и алгоритмы: пер. с англ. / М.Н. Свами, К. Тхуласираман; под ред. В.А. Горбатова. – М.: Мир, 1984. – 454 с.

35. Семенов С.Г. Анализ методов прогнозирования в телекоммуникационных сетях автоматизированных систем управления / С.Г.Семенов // Збірник наукових праць «Системи управління, навігації та зв'язку», – К.:ЦНДІ навігації і управління, – 2008.-Вип. 2(6) .- С.134-137

36. Семенов С.Г. Математическая модель процесса доставки информационных пакетов в компьютерной сети системы критического применения / С.Г.Семенов, И.В.Ильина // Науково-технічний журнал «Радіоелектронні і комп'ютерні системи» Х.:ХАІ, – 2008.-Вип. 1(28) – С.162-165

37. Семенов С.Г. Оптимизация трафика на основе сбалансированной загрузки информационно-телекоммуникационной сети // Системи обробки інформації. – Х.: ХВУ, 2004. – № 8(36). – С.206-210

38. Семенов С.Г. Математическая модель мультисервисного канала связи на основе экспоненциальной GERT-сети / С.Г. Семенов, Є.В. Мелешко, Я.В. Ілюшко // Системи озброєння і військова техніка. – Х.:ХУ ПС. – 2011. –Вип. 3(27). – С. 64-67.

39. Семенов С.Г. Математична модель системи криптографічного захисту електронних повідомлень на основі GERT-мережі / С.Г. Семенов, О.О. Сур // Системи управління, навігації та зв'язку. – К.:ЦНДІ навігації і управління. – 2012. – Том 1. Вип. 1(21). – С. 131-137

40. Семенов С.Г. Исследования вероятностно-временных характеристик мультисервисного канала связи с использованием математического аппарата GERT-сети / С.Г. Семенов, В.В. Босько, І.А. Березюк // Системи обробки інформації. – Х.: ХУ ПС. – 2012. – Том 1. Вип. 3(101). – С. 139-142.

41. Семенов С.Г. Моделирование защищенного канала связи с использованием экспоненциальной GERT-сети / С.Г. Семенов, А.А. Можаев // Информатика, математическое моделирование, экономика. – Смоленськ.: Смоленский филиал АНО ВПО ЦС РФ "Российский университет кооперации". – 2012. – Том.1. – С. 152-160.

42. Семенов С.Г. Методика математического моделирования защищенной ИТС на основе многослойной GERT-сети / С.Г. Семенов // Вісник Національного технічного університету «Харківський політехнічний інститут». – Х.:НТУ «ХПІ». – 2012. –№62 (968). – С 173-181.

43. Семенов С.Г. Защита данных в компьютеризированных управляющих системах / С.Г. Семенов, В.В. Давыдов, С.Ю. Гавриленко. – LAP Lambert Academic Publishing GmbH & Co. KG (Саарбрюккен, Германия), 2014. – 236 с.

44. Смирнов А.А. Анализ и сравнительное исследование перспективных направлений развития цифровых телекоммуникационных систем и сетей / А.А.Смирнов, В.В.Босько, Е.В.Мелешко // Системи обробки інформації. – Х.: ХУ ПС, 2008. – Вип.7(74). – С.120-123.

45. Смирнов А.А. Усовершенствование метода управления очередями в многопротокольных узлах телекоммуникационной сети / А.А.Смирнов, Е.В.Мелешко // Збірник тез та доповідей другої всеукраїнської науково-практичної конференції «Системний аналіз. Інформатика. Управління». Запоріжжя. Тези доповідей. Запоріжжя: КПУ, 2011.

46. Современные телекоммуникации. Технологии и экономика / [В.Л. Банкет, О.В. Бондаренко, П.П. Воробьенко и др.]; под ред. С.А. Довгого. – М.: Эко-Трендз, 2003. – 320 с.

47. Столлингс В. Современные компьютерные сети / Вильям Столлингс.– СПб.: Питер, 2003. – 778 с.

48. Таненбаум Э. Компьютерные сети / Эндрю Таненбаум; пер. с англ. А. Леонтьев. – СПб.: Питер, 2002. – 848 с.

49. Телекоммуникационные системы и сети: учебное пособие. В 3 томах / [В.В. Величко, Е.А. Субботин, В.П. Шувалов, А.Ф. Ярославцев]; под ред. В.П. Шувалова. – М.: Горячая линия-Телеком, 2005, т. 3 – 592 с.

50. Уолрэнд Дж. Телекоммуникационные и компьютерные сети / Дж. Уолрэнд. – М.: Постмаркет, 2001. – 480 с.

51. Хайкин С. Нейронные сети: полный курс / С. Хайкин. – М.:Вильямс, 2006. – 1103 с.

52. Шелухин О.И. Фрактальные процессы в телекоммуникациях: моногр. / О.И. Шелухин, А.М. Тенякшев, А.В. Осин – М.: Радиотехника, 2003. – 480 с.

A. Elwalid Routing and Protection in GMPLS Networks: From Shortest Paths to Optimized Designs / A. Elwalid, D. Mitra, I. Saniee, and I. Widjaja. //Journal of lightwave technology. – 2003. – №21(11), P. 2828-28-38.

53. A.B. Bagula Online Traffic Engineering: The Least Interference Optimization Algorithm / A.B. Bagula, M. Botha, and A.E Krzesinski. // IEEE Communications Society – 2004, P. 1232-1236.

54. Anees. Shaikh Evaluating the Impact of Stale Link State on Quality-of-Service Routing / Anees Shaikh, Jennifer Rexford, and Kang G. Shin. // IEEE/ACM Transactions on Networking. – 2001. – №9(2), P. 162-176.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи Network Performance Monitoring.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 204-02 від 28.12.2020 року).

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення системи Network Performance Monitoring.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи Network Performance Monitoring;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows XP/Vista/7/8/10 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows XP/Vista/7/8/10.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Delphi 10.4 Sydney.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 61 аркуш.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 22.05.2021 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист 13.06.2021 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник кваліфікаційної бакалаврської роботи

_____ Дрєєв О.М.

Програмне забезпечення системи Network Performance Monitoring

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 50

Літера: РП

Кропивницький – 2021 року

Файл проекту NetworkPerformanceMonitoring.dpr

```
program NetworkPerformanceMonitoring;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до бакалаврської роботи
на тему: Програмне забезпечення системи Network Performance Monitoring
Виконав: студент  4 курсу Хуторний Дмитро Олексійович
Керівник: Дрєєв О.М
2021 рік
}
uses
// Додавання бібліотек
  Forms, // бібліотека форм
  SysUtils, // системні функції
//підключення вікна MAIN
  frmMAIN in 'frmMAIN.pas' {Form1},
//підключення вікна Settings
  frmSettings in 'frmSettings.pas' {Form2},
//підключення вікна KEYGEN
  frmROUTER in 'frmROUTER.pas' {Form3};
// файл ресурсів
{$R *.res}
begin
try
// Створення та виведення на екран вікна заставки ПЗ
U_Splash:=TU_Sp.Create(Application);
U_Sp.Show; U_Sp.Update;
U_Sp.Label1.Caption:='Завантаження';
U_Sp.Update; U_Sp.Label2.Caption:='почекайте будь ласка…';
U_Sp.Update;   Application.HintPause:=200;
Application.HintHidePause:=7000; Application.HintShortPause:=25;
// Ініціалізація кібербезпеки та програмного забезпечення
  Application.Initialize;
// Створення головного вікна програми
  Application.CreateForm(TForm1, Form1);
// Створення вікна Form2
  Application.CreateForm(TForm2, Form2);
// Створення вікна Form3
  Application.CreateForm(TForm3, Form3);
// Створення вікна AboutBox
  Application.CreateForm(TAboutBox, AboutBox);
  finally
//звільнення вікна заставки
    U_Sp.free;
end;
// запуск
  Application.Run;
end.
```

Модуль програми A1.pas

```
unit A1;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до бакалаврської роботи
на тему: Програмне забезпечення системи Network Performance Monitoring
Виконав: студент  4 курсу Хуторний Дмитро Олексійович
Керівник: Дреєв О.М
2021 рік
}
interface
// використовуємо компоненти INDY
uses
  Classes
 ,SysUtils
{$IFDEF DMB_INDY10}
 ,IdContext
 ,IdCustomTCPServer
 ,IdGlobal
{$ELSE}
 ,IdTCPClient
 ,IdTCPServer
{$ENDIF}
 ,ModBusConsts
 ,ModbusTypes
 ,SyncObjs;

type
  TModRegisterData = array[0..MaxBlockLength] of Word;

{$IFDEF DMB_INDY10}
type
  TModBusRegisterReadEvent = procedure(const Sender: TIdContext;
    const RegNr, Count: Integer; var Data: TModRegisterData) of object;
  TModBusRegisterWriteEvent = procedure(const Sender: TIdContext;
    const RegNr, Count: Integer; const Data: TModRegisterData) of object;
  TModBusErrorEvent = procedure(const Sender: TIdContext;
    const FunctionCode: Byte; const ErrorCode: Byte) of object;
  TModBusInvalidFunctionEvent = procedure(const Sender: TIdContext;
    const FunctionCode: TModBusFunction) of object;
{$ELSE}
type
  TModBusRegisterReadEvent = procedure(const Sender: TIdPeerThread;
    const RegNr, Count: Integer; var Data: TModRegisterData) of object;
  TModBusRegisterWriteEvent = procedure(const Sender: TIdPeerThread;
    const RegNr, Count: Integer; const Data: TModRegisterData) of object;
  TModBusErrorEvent = procedure(const Sender: TIdPeerThread;
    const FunctionCode: Byte; const ErrorCode: Byte) of object;
  TModBusInvalidFunctionEvent = procedure(const Sender: TIdPeerThread;
    const FunctionCode: TModBusFunction) of object;
{$ENDIF}
type
{$IFDEF DMB_INDY10}
  TW = class(TIdCustomTCPServer)
{$ELSE}
  TW = class(TIdTCPServer)
{$ENDIF}
```

```
  private
    FOneShotConnection: Boolean;
    FLogCriticalSection: TCriticalSection;
    FLogEnabled: Boolean;
    FLogFile: String;
    FLogTimeFormat: String;
    FMaxRegister: Word;
    FMinRegister: Word;
    FOnError: TModBusErrorEvent;
    FOnInvalidFunction: TModBusInvalidFunctionEvent;
    FOnReadHoldingRegisters: TModBusRegisterReadEvent;
    FOnReadInputRegisters: TModBusRegisterReadEvent;
    FOnWriteRegisters: TModBusRegisterWriteEvent;
    FPause: Boolean;
    function GetVersion: String;
    procedure SetVersion(const Value: String);
    function BufferToHex(const ByteBuffer: array of Byte): String;
    function IsLogTimeFormatStored: Boolean;
    procedure LogByteBuffer(const LogType: String; const PeerIP: String; const
ByteBuffer: array of Byte; const Size: Integer);
  protected
  {$IFDEF DMB_INDY10}
    procedure InitComponent; override;
  {$ENDIF}
  {$IFDEF DMB_INDY10}
    procedure DoError(const AContext: TIdContext; const FunctionCode: Byte;
      const ErrorCode: Byte); virtual;
    function DoExecute(AContext: TIdContext): Boolean; override;
    procedure DoInvalidFunction(const AContext: TIdContext; const FunctionCode:
TModBusFunction); virtual;
    procedure DoReadHoldingRegisters(const AContext: TIdContext; const RegNr,
Count: Integer; var Data: TModRegisterData); virtual;
    procedure DoReadInputRegisters(const AContext: TIdContext; const RegNr,
Count: Integer; var Data: TModRegisterData); virtual;
    procedure DoWriteRegisters(const AContext: TIdContext; const RegNr, Count:
Integer; const Data: TModRegisterData); virtual;
    procedure LogExceptionBuffer(const AContext: TIdContext; const Buffer:
TModBusExceptionBuffer);
    procedure LogRequestBuffer(const AContext: TIdContext; const Buffer:
TModBusRequestBuffer; const Size: Integer);
    procedure LogResponseBuffer(const AContext: TIdContext; const Buffer:
TModBusResponseBuffer; const Size: Integer);
    procedure ReadCommand(const AContext: TIdContext);
    procedure SendError(const AContext: TIdContext; const ErrorCode: Byte;
    const ReceiveBuffer: TModBusRequestBuffer);
    procedure SendResponse(const AContext: TIdContext; const FunctionCode:
TModBusFunction;
      const ReceiveBuffer: TModBusRequestBuffer; const Data: TModRegisterData);
  {$ELSE}
procedure DoError(const Sender: TIdPeerThread; const FunctionCode: Byte;
    const ErrorCode: Byte); virtual;
    function DoExecute(AThread: TIdPeerThread): Boolean; override;
    procedure DoInvalidFunction(const Sender: TIdPeerThread; const FunctionCode:
TModBusFunction); virtual;
    procedure DoReadHoldingRegisters(const Sender: TIdPeerThread; const RegNr,
Count: Integer; var Data: TModRegisterData); virtual;
    procedure DoReadInputRegisters(const Sender: TIdPeerThread; const RegNr,
Count: Integer; var Data: TModRegisterData); virtual;
    procedure DoWriteRegisters(const Sender: TIdPeerThread; const RegNr, Count:
Integer; const Data: TModRegisterData); virtual;
```

```
    procedure LogExceptionBuffer(const AThread: TIdPeerThread; const Buffer:
TModBusExceptionBuffer);
    procedure LogRequestBuffer(const AThread: TIdPeerThread; const Buffer:
TModBusRequestBuffer; const Size: Integer);
    procedure LogResponseBuffer(const AThread: TIdPeerThread; const Buffer:
TModBusResponseBuffer; const Size: Integer);
    procedure ReadCommand(const AThread: TIdPeerThread);
    procedure SendError(const AThread: TIdPeerThread; const ErrorCode: Byte;
    const ReceiveBuffer: TModBusRequestBuffer);
    procedure SendResponse(const AThread: TIdPeerThread; const FunctionCode:
TModBusFunction;
      const ReceiveBuffer: TModBusRequestBuffer; const Data: TModRegisterData);
  {$ENDIF}
  procedure GetRegisters(const Count: Integer; const Buffer:
TModBusRequestBuffer; var Data: TModRegisterData);
  public
  {$IFNDEF DMB_INDY10}
    constructor Create(AOwner: TComponent); override;
  {$ENDIF}
  { public properties }
    property Pause: Boolean read FPause write FPause;
  published
    property DefaultPort default MB_PORT;
    property LogEnabled: Boolean read FLogEnabled write FLogEnabled default
False;
    property LogFile: String read FLogFile write FLogFile;
    property LogTimeFormat: String read FLogTimeFormat write FLogTimeFormat
stored IsLogTimeFormatStored;
    property OneShotConnection: Boolean read FOneShotConnection write
FOneShotConnection default False;
    property MaxRegister: Word read FMaxRegister write FMaxRegister default
$FFFF;
    property MinRegister: Word read FMinRegister write FMinRegister default 1;
    property Version: String read GetVersion write SetVersion stored False;
  { events }
    property OnError: TModBusErrorEvent read FOnError write FOnError;
    property OnInvalidFunction: TModBusInvalidFunctionEvent read
FOnInvalidFunction write FOnInvalidFunction;
    property OnReadHoldingRegisters: TModBusRegisterReadEvent read
FOnReadHoldingRegisters write FOnReadHoldingRegisters;
    property OnReadInputRegisters: TModBusRegisterReadEvent read
FOnReadInputRegisters write FOnReadInputRegisters;
    property OnWriteRegisters: TModBusRegisterWriteEvent read FOnWriteRegisters
write FOnWriteRegisters;
  end; { TW }
implementation
uses
  Math, Windows;

{ TW }
function TW.BufferToHex(const ByteBuffer: array of Byte): String;
var
  i: Integer;
begin
  Result := '';
  for i := Low(ByteBuffer) to High(ByteBuffer) do
    Result := Result + IntToHex(ByteBuffer[i], 2);
end;

{$IFDEF DMB_INDY10}
procedure TW.InitComponent;
```

```
{$ELSE}
constructor TW.Create(AOwner: TComponent);
{$ENDIF}
begin
{$IFDEF DMB_INDY10}
  inherited;
{$ELSE}
  inherited Create(AOwner);
{$ENDIF}
  DefaultPort := MB_PORT;
  FLogCriticalSection := TCriticalSection.Create;
  FLogEnabled := False;
  FLogFile := '';
  FLogTimeFormat := DefaultLogTimeFormat;
  FMaxRegister := $FFFF;
  FMinRegister := 1;
  FOneShotConnection := False;
  FOnError := nil;
  FOnInvalidFunction := nil;
  FOnReadHoldingRegisters := nil;
  FOnReadInputRegisters := nil;
  FOnWriteRegisters := nil;
  FPause := False;
end;

procedure TW.LogByteBuffer(const LogType: String;
  const PeerIP: String; const ByteBuffer: array of Byte; const Size: Integer);
var
  F: TextFile;
begin
  if FLogEnabled and (FLogFile <> '') then
  begin
    FLogCriticalSection.Enter;
    try
      AssignFile(F, FLogFile);
      if FileExists(FLogFile) then
        Append(F)
      else
        Rewrite(F);
      try
        WriteLn(F, FormatDateTime(FLogTimeFormat, Now)
                 ,'; ', LogType
                 ,'; ', PeerIP
                 ,'; ', IntToStr(Size)
                 ,'; ', BufferToHex(ByteBuffer));
      finally
        CloseFile(F);
      end;
    finally
      FLogCriticalSection.Leave;
    end;
  end;
end;

{$IFDEF DMB_INDY10}
procedure TW.LogExceptionBuffer(const AContext: TIdContext; const Buffer:
TModBusExceptionBuffer);
{$ELSE}
```

```
procedure TW.LogExceptionBuffer(const AThread: TIdPeerThread; const Buffer:
TModBusExceptionBuffer);
{$ENDIF}
var
  PeerIP: String;
  ByteBuffer: array of Byte;
begin
{$IFDEF DMB_INDY10}
  PeerIP := AContext.Connection.Socket.Binding.PeerIP;
{$ELSE}
  PeerIP := AThread.Connection.Socket.Binding.PeerIP;
{$ENDIF}
  SetLength(ByteBuffer, SizeOf(Buffer));
  Move(Buffer, ByteBuffer[0], SizeOf(Buffer));
  LogByteBuffer('excp', PeerIP, ByteBuffer, SizeOf(Buffer));
end;

{$IFDEF DMB_INDY10}
procedure TW.LogRequestBuffer(const AContext: TIdContext; const Buffer:
TModBusRequestBuffer; const Size: Integer);
{$ELSE}
procedure TW.LogRequestBuffer(const AThread: TIdPeerThread; const Buffer:
TModBusRequestBuffer; const Size: Integer);
{$ENDIF}
var
  PeerIP: String;
  ByteBuffer: array of Byte;
begin
{$IFDEF DMB_INDY10}
  PeerIP := AContext.Connection.Socket.Binding.PeerIP;
{$ELSE}
  PeerIP := AThread.Connection.Socket.Binding.PeerIP;
{$ENDIF}
  SetLength(ByteBuffer, SizeOf(Buffer));
  Move(Buffer, ByteBuffer[0], SizeOf(Buffer));
  LogByteBuffer('recv', PeerIP, ByteBuffer, Size);
end;

{$IFDEF DMB_INDY10}
procedure TW.LogResponseBuffer(const AContext: TIdContext; const Buffer:
TModBusResponseBuffer; const Size: Integer);
{$ELSE}
procedure TW.LogResponseBuffer(const AThread: TIdPeerThread; const Buffer:
TModBusResponseBuffer; const Size: Integer);
{$ENDIF}
var
  PeerIP: String;
  ByteBuffer: array of Byte;
begin
{$IFDEF DMB_INDY10}
  PeerIP := AContext.Connection.Socket.Binding.PeerIP;
{$ELSE}
  PeerIP := AThread.Connection.Socket.Binding.PeerIP;
{$ENDIF}
  SetLength(ByteBuffer, SizeOf(Buffer));
  Move(Buffer, ByteBuffer[0], SizeOf(Buffer));
  LogByteBuffer('sent', PeerIP, ByteBuffer, Size);
end;
```

```
{$IFDEF DMB_INDY10}
procedure TW.ReadCommand(const AContext: TIdContext);
{$ELSE}
procedure TW.ReadCommand(const AThread: TIdPeerThread);
{$ENDIF}
  function GetRegNr(const RegNr: Integer): Integer;
  begin
    Result := RegNr;
    if (RegNr < 0) then
      Result := Result + MAXWORD
    else if (RegNr > MAXWORD) then
      Result := RegNr - (MAXWORD + 1);
  end; { GetRegNr }

var
  iCount: Integer;
  iRegNr: Integer;
  ErrorCode: Byte;
  ReceiveBuffer: TModBusRequestBuffer;
  Data: TModRegisterData;
{$IFDEF DMB_INDY10}
  Buffer: TBytes;
{$ENDIF}
begin
{ Initialize all register data to 0 }
  FillChar(Data[0], SizeOf(Data), 0);
  FillChar(ReceiveBuffer, SizeOf(ReceiveBuffer), 0);
{ Read the data from the peer connection }
{$IFDEF DMB_INDY10}
  SetLength(Buffer, SizeOf(ReceiveBuffer));
  FillChar(Buffer[0], SizeOf(ReceiveBuffer), 0);
  if AContext.Binding.Readable then
    AContext.Binding.Receive(Buffer);
  Move(Buffer[0], ReceiveBuffer, Min(Length(Buffer), SizeOf(ReceiveBuffer)));
  if FLogEnabled then
    LogRequestBuffer(AContext, ReceiveBuffer,
Swap(ReceiveBuffer.Header.RecLength) + 6);
{$ELSE}
  AThread.Connection.Socket.Recv(ReceiveBuffer, SizeOf(ReceiveBuffer));
  if FLogEnabled then
    LogRequestBuffer(AThread, ReceiveBuffer,
Swap(ReceiveBuffer.Header.RecLength) + 6);
{$ENDIF}
{ Process the data }
  if ((Byte(ReceiveBuffer.FunctionCode) and $80) <> 0) then
  begin
    ErrorCode := Integer(ReceiveBuffer.MBPData[0]);
  {$IFDEF DMB_INDY10}
    DoError(AContext, ReceiveBuffer.FunctionCode and not $80, ErrorCode);
  {$ELSE}
    DoError(AThread, ReceiveBuffer.FunctionCode and not $80, ErrorCode);
  {$ENDIF}
  end
  else
  begin
     case ReceiveBuffer.FunctionCode of
      mbfReadInputRegs,
      mbfReadHoldingRegs:
        begin
```

```
        iRegNr := GetRegNr(Swap(Word((@ReceiveBuffer.MBPData[0])^)) + 1);
          iCount := Swap(Word((@ReceiveBuffer.MBPData[2])^));
          if ((iRegNr < FMinRegister) or ((iRegNr + iCount) > FMaxRegister))
then
          {$IFDEF DMB_INDY10}
            SendError(AContext, mbeIllegalRegister, ReceiveBuffer)
          {$ELSE}
            SendError(AThread, mbeIllegalRegister, ReceiveBuffer)
          {$ENDIF}
          else
          begin
          { Signal the user that data is needed }
          {$IFDEF DMB_INDY10}
            if (ReceiveBuffer.FunctionCode = mbfReadInputRegs) then
              DoReadInputRegisters(AContext, iRegNr, iCount, Data)
            else
     DoReadHoldingRegisters(AContext, iRegNr, iCount, Data);
SendResponse(AContext, ReceiveBuffer.FunctionCode, ReceiveBuffer, Data);
          {$ELSE}
            if (ReceiveBuffer.FunctionCode = mbfReadInputRegs) then
              DoReadInputRegisters(AThread, iRegNr, iCount, Data)
            else
              DoReadHoldingRegisters(AThread, iRegNr, iCount, Data);
SendResponse(AThread, ReceiveBuffer.FunctionCode, ReceiveBuffer, Data);
          {$ENDIF}
          end;
        end;
      mbfWriteOneReg:
        begin
        { Get the register number }
iRegNr := GetRegNr(Swap(Word((@ReceiveBuffer.MBPData[0])^)) + 1);
        { Get the register value }
          Data[0] := Swap(Word((@ReceiveBuffer.MBPData[2])^));
        { This function always writes one register }
          iCount := 1;
  if ((iRegNr < FMinRegister) or ((iRegNr + iCount) > FMaxRegister)) then
          {$IFDEF DMB_INDY10}
            SendError(AContext, mbeIllegalRegister, ReceiveBuffer)
          {$ELSE}
            SendError(AThread, mbeIllegalRegister, ReceiveBuffer)
          {$ENDIF}
          else
          begin
          { Send back the response to the master }
          {$IFDEF DMB_INDY10}
            SendResponse(AContext, mbfWriteOneReg, ReceiveBuffer, Data);
            DoWriteRegisters(AContext, iRegNr, iCount, Data);
          {$ELSE}
            SendResponse(AThread, mbfWriteOneReg, ReceiveBuffer, Data);
            DoWriteRegisters(AThread, iRegNr, iCount, Data);
          {$ENDIF}
          end;
        end;
      mbfWriteRegs:
        begin
iRegNr := GetRegNr(Swap(Word((@ReceiveBuffer.MBPData[0])^)) + 1);
          iCount := Swap(Word((@ReceiveBuffer.MBPData[2])^));
if ((iRegNr < FMinRegister) or ((iRegNr + iCount) > FMaxRegister)) then
          {$IFDEF DMB_INDY10}
```

```
            SendError(AContext, mbeIllegalRegister, ReceiveBuffer)
          {$ELSE}
            SendError(AThread, mbeIllegalRegister, ReceiveBuffer)
          {$ENDIF}
          else
          begin
          { Decode the contents of the Registers }
            GetRegisters(iCount, ReceiveBuffer, Data);
          { Send back the response to the master }
          {$IFDEF DMB_INDY10}
            SendResponse(AContext, mbfWriteRegs, ReceiveBuffer, Data);
            DoWriteRegisters(AContext, iRegNr, iCount, Data);
          {$ELSE}
            SendResponse(AThread, mbfWriteRegs, ReceiveBuffer, Data);
            DoWriteRegisters(AThread, iRegNr, iCount, Data);
          {$ENDIF}
          end;
        end;
    else
      if (ReceiveBuffer.FunctionCode <> 0) then
      begin
      { Illegal or unsupported function code }
      {$IFDEF DMB_INDY10}
        SendError(AContext, mbeIllegalFunction, ReceiveBuffer);
        DoInvalidFunction(AContext, ReceiveBuffer.FunctionCode);
      {$ELSE}
        SendError(AThread, mbeIllegalFunction, ReceiveBuffer);
        DoInvalidFunction(AThread, ReceiveBuffer.FunctionCode);
      {$ENDIF}
      end;
    end;
  end;
{
При необхідності: сервер завершує з'єднання, після того, як запит був оброблений
}
  if FOneShotConnection then
  {$IFDEF DMB_INDY10}
    AContext.Connection.Disconnect;
  {$ELSE}
    AThread.Connection.Disconnect;
  {$ENDIF}
end;
{$IFDEF DMB_INDY10}
procedure TW.DoError(const AContext: TIdContext;
  const FunctionCode: Byte; const ErrorCode: Byte);
{$ELSE}
procedure TW.DoError(const Sender: TIdPeerThread;
  const FunctionCode: Byte; const ErrorCode: Byte);
{$ENDIF}
begin
  if Assigned(FOnError) then
  {$IFDEF DMB_INDY10}
    FOnError(AContext, FunctionCode, ErrorCode);
  {$ELSE}
    FOnError(Sender, FunctionCode, ErrorCode);
  {$ENDIF}
end;
{$IFDEF DMB_INDY10}
function TW.DoExecute(AContext: TIdContext): Boolean;
```

```
{$ELSE}
function TW.DoExecute(AThread: TIdPeerThread): Boolean;
{$ENDIF}
begin
  Result := False;
  if not FPause then
  begin
  {$IFDEF DMB_INDY10}
    ReadCommand(AContext);
    Result := inherited DoExecute(AContext);
  {$ELSE}
    ReadCommand(AThread);
    Result := inherited DoExecute(AThread);
  {$ENDIF}
  end;
end;
{$IFDEF DMB_INDY10}
procedure TW.DoInvalidFunction(const AContext: TIdContext; const FunctionCode:
TModBusFunction);
{$ELSE}
procedure TW.DoInvalidFunction(const Sender: TIdPeerThread; const FunctionCode:
TModBusFunction);
{$ENDIF}
begin
  if Assigned(FOnInvalidFunction) then
  {$IFDEF DMB_INDY10}
    FOnInvalidFunction(AContext, FunctionCode);
  {$ELSE}
    FOnInvalidFunction(Sender, FunctionCode);
  {$ENDIF}
end;
{$IFDEF DMB_INDY10}
procedure TW.DoReadHoldingRegisters(const AContext: TIdContext; const RegNr,
Count: Integer; var Data: TModRegisterData);
{$ELSE}
procedure TW.DoReadHoldingRegisters(const Sender: TIdPeerThread; const RegNr,
Count: Integer; var Data: TModRegisterData);
{$ENDIF}
begin
  if Assigned(FOnReadHoldingRegisters) then
  {$IFDEF DMB_INDY10}
    FOnReadHoldingRegisters(AContext, RegNr, Count, Data);
  {$ELSE}
    FOnReadHoldingRegisters(Sender, RegNr, Count, Data);
  {$ENDIF}
end;
{$IFDEF DMB_INDY10}
procedure TW.DoReadInputRegisters(const AContext: TIdContext; const RegNr,
Count: Integer; var Data: TModRegisterData);
{$ELSE}
procedure TW.DoReadInputRegisters(const Sender: TIdPeerThread; const RegNr,
Count: Integer; var Data: TModRegisterData);
{$ENDIF}
begin
  if Assigned(FOnReadInputRegisters) then
  {$IFDEF DMB_INDY10}
    FOnReadInputRegisters(AContext, RegNr, Count, Data);
  {$ELSE}
    FOnReadInputRegisters(Sender, RegNr, Count, Data);
```

```
  {$ENDIF}
end;
{$IFDEF DMB_INDY10}
procedure TW.DoWriteRegisters(const AContext: TIdContext; const RegNr, Count:
Integer; const Data: TModRegisterData);
{$ELSE}
procedure TW.DoWriteRegisters(const Sender: TIdPeerThread; const RegNr, Count:
Integer; const Data: TModRegisterData);
{$ENDIF}
begin
  if Assigned(FOnWriteRegisters) then
  {$IFDEF DMB_INDY10}
    FOnWriteRegisters(AContext, RegNr, Count, Data);
  {$ELSE}
    FOnWriteRegisters(Sender, RegNr, Count, Data);
  {$ENDIF}
end;
procedure TW.GetRegisters(const Count: Integer; const Buffer:
TModBusRequestBuffer; var Data: TModRegisterData);
var
  WPtr: ^Word;
  i: Integer;
begin
  WPtr := @Buffer.MbpData[5];
  for i := 0 to (Count - 1) do
  begin
    Data[i] := Swap(WPtr^);
    Inc(WPtr);
  end;
end;

{$IFDEF DMB_INDY10}
procedure TW.SendError(const AContext: TIdContext;
  const ErrorCode: Byte; const ReceiveBuffer: TModBusRequestBuffer);
{$ELSE}
procedure TW.SendError(const AThread: TIdPeerThread;
  const ErrorCode: Byte; const ReceiveBuffer: TModBusRequestBuffer);
{$ENDIF}
var
  SendBuffer: TModBusExceptionBuffer;
{$IFDEF DMB_INDY10}
  Buffer: TIdBytes;
{$ENDIF}
begin
  if Active then
  begin
    SendBuffer.Header := ReceiveBuffer.Header;
    SendBuffer.ExceptionFunction := ReceiveBuffer.FunctionCode or $80;
    SendBuffer.ExceptionCode := ErrorCode;
    SendBuffer.Header.RecLength := Swap(3);

  {$IFDEF DMB_INDY10}
    Buffer := RawToBytes(SendBuffer, SizeOf(SendBuffer));
    AContext.Connection.Socket.WriteDirect(Buffer);
    if FLogEnabled then
      LogExceptionBuffer(AContext, SendBuffer);
  {$ELSE}
    AThread.Connection.Socket.Send(SendBuffer, SizeOf(SendBuffer));
    if FLogEnabled then
```

```
      LogExceptionBuffer(AThread, SendBuffer);
  {$ENDIF}
  end;
end;

{$IFDEF DMB_INDY10}
procedure TW.SendResponse(const AContext: TIdContext;
  const FunctionCode: TModBusFunction; const ReceiveBuffer:
TModBusRequestBuffer;
  const Data: TModRegisterData);
{$ELSE}
procedure TW.SendResponse(const AThread: TIdPeerThread;
  const FunctionCode: TModBusFunction; const ReceiveBuffer:
TModBusRequestBuffer;
  const Data: TModRegisterData);
{$ENDIF}
var
  SendBuffer: TModBusResponseBuffer;
  iBytesToWrite: Byte;
  L, i: Integer;
  WordPtr: ^Word;
{$IFDEF DMB_INDY10}
  Buffer: TIdBytes;
{$ENDIF}
begin
  if Active then
  begin
    FillChar(SendBuffer, SizeOf(SendBuffer), 0);
    SendBuffer.Header.TransactionID := ReceiveBuffer.Header.TransactionID;
    SendBuffer.Header.ProtocolID := ReceiveBuffer.Header.ProtocolID;
    SendBuffer.Header.UnitID := ReceiveBuffer.Header.UnitID;
    SendBuffer.FunctionCode := ReceiveBuffer.FunctionCode;
    SendBuffer.Header.RecLength := Swap(0);

    case FunctionCode of
      mbfWriteRegs:
        begin
          SendBuffer.MBPData[0] := ReceiveBuffer.MBPData[0];
          SendBuffer.MBPData[1] := ReceiveBuffer.MBPData[1];
          SendBuffer.MBPData[2] := ReceiveBuffer.MBPData[2];
          SendBuffer.MBPData[3] := ReceiveBuffer.MBPData[3];
          SendBuffer.Header.RecLength := Swap(6);
        end;
      mbfWriteOneReg:
        begin
          SendBuffer.MBPData[0] := ReceiveBuffer.MBPData[0];
          SendBuffer.MBPData[1] := ReceiveBuffer.MBPData[1];
          SendBuffer.MBPData[2] := ReceiveBuffer.MBPData[2];
          SendBuffer.MBPData[3] := ReceiveBuffer.MBPData[3];
          SendBuffer.Header.RecLength := Swap(6);
        end;
      mbfReadInputRegs,
      mbfReadHoldingRegs:
        begin
          L := Swap(Word((@ReceiveBuffer.MBPData[2])^));
          if (L <= MaxBlockLength) then
          begin
            iBytesToWrite := Byte(L shl 1);
            SendBuffer.MBPData[0] := iBytesToWrite;
```

```
            WordPtr := @SendBuffer.MBPData[1];
            for i := 0 to (L - 1) do
            begin
              WordPtr^ := Swap(Data[i]);
              Inc(WordPtr);
            end;
            SendBuffer.Header.RecLength := Swap(3 + iBytesToWrite);
          end;
        end;
    end;
  {$IFDEF DMB_INDY10}
Buffer := RawToBytes(SendBuffer, Swap(SendBuffer.Header.RecLength) + 6);
    AContext.Connection.Socket.WriteDirect(Buffer);
    if FLogEnabled then
      LogResponseBuffer(AContext, SendBuffer, Swap(SendBuffer.Header.RecLength)
+ 6);
  {$ELSE}
    AThread.Connection.Socket.Send(SendBuffer, Swap(SendBuffer.Header.RecLength)
+ 6);
    if FLogEnabled then
      LogResponseBuffer(AThread, SendBuffer, Swap(SendBuffer.Header.RecLength) +
6);
  {$ENDIF}
  end;
end;

function TW.GetVersion: String;
begin
  Result := DMB_VERSION;
end;

function TW.IsLogTimeFormatStored: Boolean;
begin
  Result := (FLogTimeFormat <> DefaultLogTimeFormat);
end;

procedure TW.SetCount(const Value: Cardinal);
var
 THX, TDX: LongInt;
asm
   test  edx, edx
   jg    @@10
   mov   [eax].FCount, 0
   lea   edx, [eax].FHandle
   call  TW.FreeMem
   jmp   @@30
@@10:
   cmp   edx, [eax].FCount
   je    @@30
   push  esi
   push  edi
   mov   esi, eax
   mov   edi, edx
   lea   ecx, thx
   call  TW.AllocMem
   mov   tdx, eax
   mov   ecx, [esi].FCount
   mov   edx, edi
   cmp   edx, ecx
```

```
    jl    @@20
    mov   edx, ecx
@@20:
    mov   eax, edx
    mov   edx, [esi].FItemSize
    imul  edx
    mov   ecx, eax
    mov   edx, tdx
    mov   eax, [esi].FData
    call  MoveMem
    mov   eax, tdx
    mov   [esi].FData, eax
    lea   edx, [esi].FHandle
    mov   eax, esi
    call  TW.FreeMem
    mov   eax, thx
    mov   [esi].FHandle, eax
    mov   [esi].FCount, edi
    mov   eax, esi
    call  TW.DoSizeChanged
    pop   edi
    pop   esi
@@30:
end;
procedure TW.Swap(Index1, Index2: Cardinal);
var
    thx, tdx: LongInt;
asm
    push  esi
    push  edi
    push  ebx
    mov   ebx, eax
    mov   esi, edx
    mov   edi, ecx
    mov   edx, 1
    lea   ecx, thx
    call  TW.AllocMem
    mov   tdx, eax
    mov   eax, ebx
    mov   edx, esi
    call  TW.GetItemPtr
    mov   edx, esi
    test  eax, eax
    jz    @@10
    mov   esi, eax
    mov   edx, tdx
    mov   ecx, [ebx].FItemSize
    call  MoveMem
    mov   eax, ebx
    mov   edx, edi
    call  TW.GetItemPtr
    mov   edx, edi
    test  eax, eax
    jz    @@10
    mov   edi, eax
    mov   edx, esi
    mov   ecx, [ebx].FItemSize
    call  MoveMem
    mov   eax, tdx
```

```
   mov   edx, edi
   mov   ecx, [ebx].FItemSize
   call  MoveMem
   mov   eax, ebx
   lea   edx, thx
   call  TW.FreeMem
   pop   ebx
   pop   edi
   pop   esi
   jmp   @@20
@@10:
   mov   eax, ebx
   push  eax
   push  edx
   lea   edx, thx
   call  TW.FreeMem
   pop   edx
   pop   eax
   pop   ebx
   pop   edi
   pop   esi
   call  TW.Error
@@20:
end;
procedure TW.Trim(Count: Cardinal);
asm
   mov    ecx, edx
   mov    edx, [eax].FCount
   sub    edx, ecx
   call   TW.SetCount
end;

procedure TW.SetCount(const Value: Cardinal);
begin
 SetCount(Value);
end;
constructor TW.Create(ADataName: TString; Backup: Boolean);
begin
 FStatus:=fsWriting;
 inherited Create;
 FDataName:=ADataName;
 if Backup and DataExists(FDataName) then CreateBackup;
 FHandle:=CreateData(PChar(FDataName), GENERIC_WRITE, 0, nil, CREATE_ALWAYS,
  DATA_ATTRIBUTE_NORMAL, 0);
 if FHandle = INVALID_HANDLE_VALUE then Error(GetLastError);
end;
procedure TW.CreateBackup;
var
 BackupName: TString;
 Ext: TString;
begin
 BackupName:=FDataName;
 Ext:=ExtractDataExt(BackupName);
 BackupName:=TrailTrim(BackupName, Length(Ext));
 Delete(Ext, 1, 1);
 BackupName:=BackupName+'.~'+Ext;
 if DataExists(BackupName) then DeleteData(BackupName);
 if not RenameData(FDataName, BackupName) then Error(GetLastError)
end;
```

```
class function TW.EncodeDateTime(Year, Month, Day, Hour, Min,
                                        Sec: Word): TDataTime;
var
 LT: TDataTime;
 ST: TSystemTime;
begin
 ST.wYear:=Year;
 ST.wMonth:=Month;
 ST.wDayOfWeek:=0;
 ST.wDay:=Day;
 ST.wHour:=Hour;
 ST.wMinute:=Min;
 ST.wSecond:=Sec;
 ST.wMilliseconds:=0;
 SystemTimeToDataTime(ST, LT);
 LocalDataTimeToDataTime(LT, Result);
end;
destructor TW.Destroy;
begin
 CloseHandle(FHandle);
 inherited;
end;
procedure TW.Error(Code: Integer);
const
 strDataStatus : array[TDataStatus] of TString = (SDataReading, SDataWriting);
begin
 if Code<>0 then raise EDataError.CreateFmt(SDataError,
  [strDataStatus[FStatus], FDataName, GetErrorMessage(Code)]);
end;
class procedure TW.DecodeDateTime(const DateTime: TDataTime; Year,
                                    Month, Day, Hour, Min, Sec: PWord);
var
 LT: TDataTime;
 ST: TSystemTime;
begin
 DataTimeToLocalDataTime(DateTime, LT);
 DataTimeToSystemTime(LT, ST);
 SetWordValue(Year, ST.wYear);
 SetWordValue(Month, ST.wMonth);
 SetWordValue(Day, ST.wDay);
 SetWordValue(Hour, ST.wHour);
 SetWordValue(Min, ST.wMinute);
 SetWordValue(Sec, ST.wSecond);
end;
function TW.GetAttributes: LongInt;
begin
 Result:=GetDataAttributes(PChar(FDataName));
 if Result = LongInt($FFFFFFFF) then Error(GetLastError);
end;
function TW.GetCreationTime: TDataTime;
begin
 if not GetDataTime(FHandle, @Result, nil, nil) then Error(GetLastError);
end;
function TW.GetLastAccessTime: TDataTime;
begin
 if not GetDataTime(FHandle, nil, @Result, nil) then Error(GetLastError);
end;
function TW.GetLastWriteTime: TDataTime;
begin
```

```
 if not GetDataTime(FHandle, nil, nil, @Result) then Error(GetLastError);
end;
function TW.GetSize: Integer;
begin
 Result:=GetDataSize(FHandle, nil);
 if Result = -1 then Error(GetLastError);
end;
constructor TW.Open(ADataName: TString);
begin
 inherited Create;
 FStatus:=fsReading;
 FDataName:=ADataName;
 FHandle:=CreateData(PChar(FDataName), GENERIC_READ, 0, nil, OPEN_EXISTING,
  DATA_ATTRIBUTE_NORMAL, 0);
 if FHandle = INVALID_HANDLE_VALUE then Error(GetLastError);
end;
procedure TW.Read(var Buffer; Size: Integer);
begin
 if FStatus = fsReading then begin
  if not ReadData(FHandle, Buffer, Cardinal(Size), FDummy, nil)
   then Error(GetLastError);
 end;
end;
procedure TW.Seek(Position: Integer);
begin
 SetDataPointer(FHandle, Position, nil, DATA_BEGIN);
 Error(GetLastError);
end;
procedure TW.SetAttributes(const Value: LongInt);
begin
 if not SetDataAttributes(PChar(FDataName), Value) then Error(GetLastError);
end;
procedure TW.SetCreationTime(const Value: TDataTime);
begin
 if not SetDataTime(FHandle, @Value, nil, nil) then Error(GetLastError);
end;
procedure TW.SetLastAccessTime(const Value: TDataTime);
begin
 if not SetDataTime(FHandle, nil, @Value, nil) then Error(GetLastError);
end;
procedure TW.SetLastWriteTime(const Value: TDataTime);
begin
 if not SetDataTime(FHandle, nil, nil, @Value) then Error(GetLastError);
end;
procedure TW.UserError(Code: Integer);
begin
 Error(Code);
end;
procedure TW.Write(const Buffer; Size: Integer);
begin
 if FStatus = fsWriting then begin
  if not WriteData(FHandle, Buffer, Cardinal(Size), FDummy, nil)
   then Error(GetLastError);
 end; end;
procedure TW.Close;
begin
 Free;
end;
```

```
constructor TW.Create(ADataName: TString; Backup: Boolean);
begin
 FStatus:=fsWriting;
 inherited Create;
 FDataName:=ADataName;
 if Backup and DataExists(FDataName) then CreateBackup;
 FHandle:=CreateData(PChar(FDataName), GENERIC_WRITE, 0, nil, CREATE_ALWAYS,
  DATA_ATTRIBUTE_NORMAL, 0);
 if FHandle = INVALID_HANDLE_VALUE then Error(GetLastError);
end;

procedure TW.CreateBackup;
var
 BackupName: TString;
 Ext: TString;
begin
 BackupName:=FDataName;
 Ext:=ExtractDataExt(BackupName);
 BackupName:=TrailTrim(BackupName, Length(Ext));
 Delete(Ext, 1, 1);
 BackupName:=BackupName+'.~'+Ext;
 if DataExists(BackupName) then DeleteData(BackupName);
 if not RenameData(FDataName, BackupName) then Error(GetLastError)
end;

class procedure TW.DecodeDateTime(const DateTime: TDataTime; Year,
  Month, Day, Hour, Min, Sec: PWord);
var
 LT: TDataTime;
 ST: TSystemTime;
begin
 DataTimeToLocalDataTime(DateTime, LT);
 DataTimeToSystemTime(LT, ST);
 SetWordValue(Year, ST.wYear);
 SetWordValue(Month, ST.wMonth);
 SetWordValue(Day, ST.wDay);
 SetWordValue(Hour, ST.wHour);
 SetWordValue(Min, ST.wMinute);
 SetWordValue(Sec, ST.wSecond);
end;

destructor TW.Destroy;
begin
 CloseHandle(FHandle);
 inherited;
 end;

class function TW.EncodeDateTime(Year, Month, Day, Hout, Min,
                                        Sec: Word): TDataTime;
var
 LT: TDataTime;
 ST: TSystemTime;
begin
 ST.wYear:=Year;
 ST.wMonth:=Month;
 ST.wDayOfWeek:=0;
 ST.wDay:=Day;
 ST.wHour:=Hour;
 ST.wMinute:=Min;
```

```
 ST.wSecond:=Sec;
 ST.wMilliseconds:=0;
 SystemTimeToDataTime(ST, LT);
 LocalDataTimeToDataTime(LT, Result);
end;

procedure TW.Error(Code: Integer);
const
 strDataStatus : array[TDataStatus] of TString = (SDataReading, SDataWriting);
begin
 if Code<>0 then raise EDataError.CreateFmt(SDataError,
  [strDataStatus[FStatus], FDataName, GetErrorMessage(Code)]);
end;

function TW.GetAttributes: LongInt;
begin
 Result:=GetDataAttributes(PChar(FDataName));
 if Result = LongInt($FFFFFFFF) then Error(GetLastError);
end;

function TW.GetCreationTime: TDataTime;
begin
 if not GetDataTime(FHandle, @Result, nil, nil) then Error(GetLastError);
end;
end;

function TW.GetLastAccessTime: TDataTime;
begin
 if not GetDataTime(FHandle, nil, @Result, nil) then Error(GetLastError);
end;

function TW.GetLastWriteTime: TDataTime;
begin
 if not GetDataTime(FHandle, nil, nil, @Result) then Error(GetLastError);
end;

constructor TW.Open(ADataName: TString);
begin
 inherited Create;
 FStatus:=fsReading;
 FDataName:=ADataName;
 FHandle:=CreateData(PChar(FDataName), GENERIC_READ, 0, nil, OPEN_EXISTING,
  DATA_ATTRIBUTE_NORMAL, 0);
 if FHandle = INVALID_HANDLE_VALUE then Error(GetLastError);
end;

function TW.Read(var Buffer; Count: Integer): LongInt;
begin
 if FStatus = fsReading then begin
  if not ReadData(FHandle, Buffer, Cardinal(Count), LongWord(Result), nil)
   then Error(GetLastError);
 end;
end;

function TW.Seek(Offset: Integer; Origin: Word): LongInt;
begin
 Result:=SetDataPointer(FHandle, Position, nil, Origin);
 Error(GetLastError);
end;
```

```
procedure TW.SetAttributes(const Value: LongInt);
begin
 if not SetDataAttributes(PChar(FDataName), Value) then Error(GetLastError);
end;

procedure TW.SetCreationTime(const Value: TDataTime);
begin
 if not SetDataTime(FHandle, @Value, nil, nil) then Error(GetLastError);
end;

procedure TW.SetLastAccessTime(const Value: TDataTime);
begin
 if not SetDataTime(FHandle, nil, @Value, nil) then Error(GetLastError);
end;

procedure TW.SetLastWriteTime(const Value: TDataTime);
begin
 if not SetDataTime(FHandle, nil, nil, @Value) then Error(GetLastError);
end;

procedure TW.SetSize(NewSize: LongInt);
begin
 raise EDataError.Create(SCannotSetSize);
end;

procedure TW.UserError(Code: Integer);
begin
 Error(Code);
end;

function TW.Write(const Buffer; Count: Integer): LongInt;
begin
 if FStatus = fsWriting then begin
  if not WriteData(FHandle, Buffer, Cardinal(Count), LongWord(Result), nil)
   then Error(GetLastError);
 end;
end;

constructor TW.Create(AColCount, ARowCount, AItemSize: Integer);
begin
 inherited Create;
 FItemSize:=AItemSize;
 FRows:=TMatrixRows.Create(Self);
 RowCount:=ARowCount;
 ColCount:=AColCount;
end;

function TW.CreateRow: TMatrixRow;
begin
 Result:=TMatrixRow.Create(ColCount, Self);
end;

procedure TW.DeleteCol(Index: Integer);
begin
 if Inside(Index, 0, FRows.Width - 1)
  then FRows.DeleteCol(Index)
  else raise EMatrixError.CreateFmt(SColIndexOutOfRange, [Index]);
end;
```

```
procedure TW.DeleteRow(Index: Integer);
begin
 if Inside(Index, 0, FRows.Count - 1) then begin
  FRows[Index].Free;
  FRows.Delete(Index);
 end else raise EMatrixError.CreateFmt(SRowIndexOutOfRange, [Index]);
end;

destructor TW.Destroy;
begin
 FRows.Free;
 inherited;
end;

function TW.ForEachRow(Tag: Integer;
  ForEachRowFunc: TForEachFunc): Integer;
begin
 Result:=FRows.ForEach(Tag, ForEachRowFunc);
end;

function TW.GetColCount: Integer;
begin
 Result:=FRows.Width;
end;

procedure TW.GetItem(ACol, ARow: Integer; out Item);
begin
 if Inside(ARow, 0, FRows.Count - 1) and Inside(ACol, 0, FRows.FWidth-1)
  then FRows[ARow].GetItem(ACol, Item)
  else raise EMatrixError.CreateFmt(SIndicesOutOfRange, [ACol, ARow]);
end;

function TW.GetRow(Index: Integer): TMatrixRow;
begin
 Result:=FRows[Index];
end;

function TW.GetRowCount: Integer;
begin
 Result:=FRows.Count;
end;

procedure TW.InsertCol(Index: Integer);
begin
 if Inside(Index, 0, FRows.Width)
  then FRows.InsertCol(Index)
  else raise EMatrixError.CreateFmt(SColIndexOutOfRange, [Index]);
end;

procedure TW.InsertRow(Index: Integer);
var
 Temp: TMatrixRow;
begin
 if Inside(Index, 0, FRows.Count) then begin
  Temp:=CreateRow;
  FRows.InsertItem(Index, Temp);
 end else raise EMatrixError.CreateFmt(SRowIndexOutOfRange, [Index]);
end;
```

```
procedure TW.PutItem(ACol, ARow: Integer; const Item);
begin
if Inside(ARow, 0, FRows.Count - 1) and Inside(ACol, 0, FRows.FWidth-1)
  then FRows[ARow].PutItem(ACol, Item)
  else raise EMatrixError.CreateFmt(SIndicesOutOfRange, [ACol, ARow]);
end;

procedure TW.SetColCount(const Value: Integer);
begin
 FRows.Width:=Value;
end;

procedure TW.SetRowCount(const Value: Integer);
var
 OldCount: Integer;
 i: Integer;
begin
 OldCount:=RowCount;
 if OldCount < Value then begin
  for i:=OldCount+1 to Value do InsertRow(RowCount);
 end else if OldCount > Value then begin
  for i:=OldCount-1 downto Value do DeleteRow(RowCount-1);
 end;
end;
end.
```

Модуль програми A2.pas

```
unit A2;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до бакалаврської роботи
на тему: Програмне забезпечення системи Network Performance Monitoring
Виконав: студент  4 курсу Хуторний Дмитро Олексійович
Керівник: Дреєв О.М
2021 рік
}
interface
// опис
// Додавання бібліотек що будуть використовуватись
uses Windows, ShellAPI, ShlObj, ActiveX, ComObj, Dim, Classes, SysUtils,
WinSock;
// Опис типів даних та класів
type
  ComputerFound = class (Exception);
  ECannotFindNetwork = class (Exception);

  TStringObject = class (TObject) // новий клас
  private
    FValue: TString;
    FTag: Integer;
    FData: Pointer;
    FRefObj: TObject;
    procedure SetValue(const Value: TString);
    procedure SetData(const Value: Pointer);
    procedure SetRefObj(const Value: TObject);
    procedure SetTag(const Value: Integer);
  public
    property Value: TString read FValue write SetValue;
    property RefObj: TObject read FRefObj write SetRefObj;
    property Tag: Integer read FTag write SetTag;
    property Data: Pointer read FData write SetData;
  end;
  TStringObjectArray = class (TDynamicArray) // новий клас
  private
    function GetObject(Index: Integer): TStringObject;
    function GetData(Index: Integer): Pointer;
    function GetRefObj(Index: Integer): TObject;
    function GetTag(Index: Integer): Integer;
    function GetValue(Index: Integer): TString;
    procedure SetData(Index: Integer; const Value: Pointer);
    procedure SetRefObj(Index: Integer; const Value: TObject);
    procedure SetTag(Index: Integer; const Value: Integer);
    procedure SetValue(Index: Integer; const Value: TString);
    function FreeObject(Index: Integer; var Obj: TStringObject): Integer;
    procedure FreeItem(Index: Integer);
    procedure CreateItem(Index: Integer);
  protected
    procedure SetCount(const NewCount: Cardinal); override;
  public
    function Add: Integer; override;
    procedure Insert(Index: Integer); override;
    procedure Delete(Index: Integer); override;
    function AddItem(const Item): Integer; override;
```

```
    procedure InsertItem(Index: Integer; const Item); override;
    procedure DeleteItem(Index: Integer; out Item); override;
    property Value[Index: Integer]: TString read GetValue write SetValue;
default;
    property RefObj[Index: Integer]: TObject read GetRefObj write SetRefObj;
    property Tag[Index: Integer]: Integer read GetTag write SetTag;
    property Data[Index: Integer]: Pointer read GetData write SetData;
    constructor Create;
    destructor Destroy; override;
  end;

  TStringObjectList = class (TStrings) // новий клас
  private
    FArray: TStringObjectArray;
    function GetData(Index: Integer): Pointer;
    function GetTag(Index: Integer): Integer;
    procedure SetData(Index: Integer; const Value: Pointer);
    procedure SetTag(Index: Integer; const Value: Integer);
  protected
    function Get(Index: Integer): string; override;
    function GetCount: Integer; override;
    function GetObject(Index: Integer): TObject; override;
    procedure Put(Index: Integer; const S: string); override;
    procedure PutObject(Index: Integer; AObject: TObject); override;

  public
    property Data[Index: Integer]: Pointer read GetData write SetData;
    property Tag[Index: Integer]: Integer read GetTag write SetTag;
    function Add(const S: string): Integer; override;
    procedure Clear; override;
    procedure Delete(Index: Integer); override;
    procedure Exchange(Index1, Index2: Integer); override;
    procedure Insert(Index: Integer; const S: string); override;
    constructor Create;
    destructor Destroy; override;
  end;

{TNetwork - клас списку всіх
 комп'ютерів в робочій групі.
 Успадкований  від TStrings і повністю сумісний з усіма
 класами списків рядків. Об'єкти цього класу запіиваются
 у властивості Objects і Workgroups об'єктів
 класу TNetworkNeiborhood}

 TNetwork = class (TStringObjectList);

  TNetworkNeighborhood = class (TStringObjectList) // новий клас
  private
    function CreatePIDL(Size: Integer): PItemIDList;
    procedure DisposePIDL(ID: PItemIDList);
    function NextPIDL(IDList: PItemIDList): PItemIDList;
    function GetPIDLSize(IDList: PItemIDList): Integer;
    function CopyPIDL(IDList: PItemIDList): PItemIDList;
    procedure StripLastID(IDList: PItemIDList);
    function GetPrevPIDL(PIDL: PItemIDList): PItemIDList;
    class function GetDisplayName(ShellFolder: IShellFolder; PIDL: PItemIDList):
TString;
    function OriginFolder:  IShellFolder;
    function OriginFolderNT: IShellFolder;
```

```
    class function EnumObjects(ShellFolder: IShellFolder): IEnumIDList;
    class procedure ParseFolder(Folder: IShellFolder; Items: TStringObjectList;
StorePIDLs: Boolean = False);
    class procedure ParseFolderEx(Folder: IShellFolder; Items: TStrings);
    function FreeRefObj(Index: Integer; var Obj: TStringObject): Integer;
    function GetWorkgroup(Name: TString): TNetwork;
  public
    procedure Refresh;
    property Workgroup[Name: TString]: TNetwork read
                                       GetWorkgroup;
    function FindComputer(Name: TString): TString;
    procedure ListComputers(Strings: TStrings);
    procedure ListNetwork(Strings: TStrings);
    function Add(const S: string): Integer; override;
    procedure Clear; override;
    procedure Delete(Index: Integer); override;
    procedure Insert(Index: Integer; const S: string); override;
    constructor Create;
  end;
function GetIPAddress(NetworkName: TString): TString;
procedure GetIPAddresses(Network: TNetworkNeighborhood; List: TStrings);
function EnumSharedResources(ComputerName: TString; List: TStrings): Boolean;

implementation // модуль

uses DimConst; // Додавання бібліотек що будуть використовуватись

// Реалізація методу класу
{ TStringObject }
procedure TStringObject.SetData(const Value: Pointer);
begin
  FData := Value;
end;
procedure TStringObject.SetRefObj(const Value: TObject);
begin
  FRefObj := Value;
end;
procedure TStringObject.SetTag(const Value: Integer);
begin
  FTag := Value;
end;

procedure TStringObject.SetValue(const Value: TString);
begin
  FValue := Value;
end;
// Реалізація методу класу
{ TStringObjectArray }
function TStringObjectArray.Add: Integer;
begin
 Result:=inherited Add;
 CreateItem(Result);
end;
function TStringObjectArray.AddItem(const Item): Integer;
begin
 Result:=Add;
end;
constructor TStringObjectArray.Create;
begin
```

```
 inherited Create(0, SizeOf(TStringObject));
end;
procedure TStringObjectArray.CreateItem(Index: Integer);
var
// об’ява типів даних що будуть використовуватися
 P: ^TStringObject;
begin
 P:=GetItemPtr(Index);
 P^:=TStringObject.Create;
end;

procedure TStringObjectArray.Delete(Index: Integer);
begin
 FreeItem(Index);
 inherited;
end;

procedure TStringObjectArray.DeleteItem(Index: Integer; out Item);
begin
 Delete(Index);
end;

destructor TStringObjectArray.Destroy;
begin
 ForEach(Integer(Self), @TStringObjectArray.FreeObject);
 inherited;
end;

procedure TStringObjectArray.FreeItem(Index: Integer);
var
// об’ява типів даних що будуть використовуватися
 P: ^TStringObject;
begin
 P:=GetItemPtr(Index);
 FreeAndNil(P^);
end;

function TStringObjectArray.FreeObject(Index: Integer;
  var Obj: TStringObject): Integer;
begin
 FreeAndNil(Obj);
 Result:=0;
end;

function TStringObjectArray.GetData(Index: Integer): Pointer;
begin
 Result:=GetObject(Index).Data;
end;
function TStringObjectArray.GetObject(Index: Integer): TStringObject;
begin
 GetItem(Index, Result);
end;
function TStringObjectArray.GetRefObj(Index: Integer): TObject;
begin
 Result:=GetObject(Index).RefObj;
end;

function TStringObjectArray.GetTag(Index: Integer): Integer;
begin
```

```
 Result:=GetObject(Index).Tag;
end;

function TStringObjectArray.GetValue(Index: Integer): TString;
begin
 Result:=GetObject(Index).Value;
end;

procedure TStringObjectArray.Insert(Index: Integer);
begin
 inherited;
 CreateItem(Index);
end;

procedure TStringObjectArray.InsertItem(Index: Integer; const Item);
begin
 Insert(Index);
end;

procedure TStringObjectArray.SetCount(const NewCount: Cardinal);
var
// об'ява типів даних що будуть використовуватися
 i, OldCount: Integer;
begin
 OldCount:=Count;
 if NewCount > Count then begin
  inherited SetCount(NewCount);
  for i:=OldCount to NewCount - 1 do CreateItem(i);
 end else if NewCount < Count then begin
  for i:=NewCount to OldCount - 1 do FreeItem(i);
  inherited SetCount(NewCount);
 end;
end;

procedure TStringObjectArray.SetData(Index: Integer; const Value: Pointer);
begin
 GetObject(Index).Data:=Value;
end;

procedure TStringObjectArray.SetRefObj(Index: Integer;
  const Value: TObject);
begin
 GetObject(Index).RefObj:=Value;
end;

procedure TStringObjectArray.SetTag(Index: Integer; const Value: Integer);
begin
 GetObject(Index).Tag:=Value;
end;

procedure TStringObjectArray.SetValue(Index: Integer; const Value: TString);
begin
 GetObject(Index).Value:=Value;
end;
// Реалізація методу класу
{ TStringObjectList }
function TStringObjectList.Add(const S: string): Integer;
begin
 Result:=FArray.Add;
```

```
 FArray.Value[Result]:=S;
end;

procedure TStringObjectList.Clear;
begin
 FArray.Count:=0;
end;

constructor TStringObjectList.Create;
begin
 inherited Create;
 FArray:=TStringObjectArray.Create;
end;

procedure TStringObjectList.Delete(Index: Integer);
begin
 FArray.Delete(Index);
end;
destructor TStringObjectList.Destroy;
begin
 FArray.Free;
 inherited;
end;
procedure TStringObjectList.Exchange(Index1, Index2: Integer);
begin
 FArray.Swap(Index1, Index2);
end;
function TStringObjectList.Get(Index: Integer): string;
begin
 Result:=FArray.Value[Index];
end;

function TStringObjectList.GetCount: Integer;
begin
 Result:=FArray.Count;
end;

function TStringObjectList.GetData(Index: Integer): Pointer;
begin
 Result:=FArray.Data[Index];
end;

function TStringObjectList.GetObject(Index: Integer): TObject;
begin
 Result:=FArray.RefObj[Index];
end;

function TStringObjectList.GetTag(Index: Integer): Integer;
begin
 Result:=FArray.Tag[Index];
end;

procedure TStringObjectList.Insert(Index: Integer; const S: string);
begin
 FArray.Insert(Index);
 FArray.Value[Index]:=S;
end;

procedure TStringObjectList.Put(Index: Integer; const S: string);
```

```
begin
 FArray.Value[Index]:=S;
end;

procedure TStringObjectList.PutObject(Index: Integer; AObject: TObject);
begin
 FArray.RefObj[Index]:=AObject;
end;
procedure TStringObjectList.SetData(Index: Integer; const Value: Pointer);
begin
 FArray.Data[Index]:=Value;
end;

procedure TStringObjectList.SetTag(Index: Integer; const Value: Integer);
begin
 FArray.Tag[Index]:=Value;
end;

// Реалізація методу класу
{ TNetworkNeighborhood }
function TNetworkN.Add(const S: string): Integer;
begin
 Result:=inherited Add(S);
 Objects[Result]:=TNetwork.Create;
end;

procedure TNetworkN.Clear;
begin
 FArray.ForEach(Integer(Self), @TNetworkN.FreeRefObj);
 inherited;
end;

function TNetworkN.CopyPIDL(IDList: PItemIDList): PItemIDList;
var
// об’ява типів даних що будуть використовуватися
 Size: Integer;
begin
 Size := GetPIDLSize(IDList);
 Result := CreatePIDL(Size);
 if Assigned(Result) then CopyMemory(Result, IDList, Size);
end;

constructor TNetworkN.Create;
begin
 inherited Create;
 Refresh;
end;

function TNetworkN.CreatePIDL(Size: Integer): PItemIDList;
var
// об’ява типів даних що будуть використовуватися
 Malloc: IMalloc;
 HR: HResult;
begin
 Result := nil;
 HR := SHGetMalloc(Malloc);
 if Failed(HR) then  Exit;
 try
  Result := Malloc.Alloc(Size);
```

```
  if Assigned(Result) then FillChar(Result^, Size, 0);
 finally
 end;
end;

procedure TNetworkN.Delete(Index: Integer);
begin
end;

procedure TNetworkN.DisposePIDL(ID: PItemIDList);
var
// об’ява типів даних що будуть використовуватися
 Malloc: IMalloc;
begin
 if ID = nil then Exit;
 OLECheck(SHGetMalloc(Malloc));
 Malloc.Free(ID);
end;

class function TNetworkN.EnumObjects(
  ShellFolder: IShellFolder): IEnumIDList;
const
 Flags = SHCONTF_FOLDERS or SHCONTF_NONFOLDERS or SHCONTF_INCLUDEHIDDEN;
begin
 ShellFolder.EnumObjects(0, Flags, Result);
end;

function TNetworkN.FindComputer(Name: TString): TString;
var
// об’ява типів даних що будуть використовуватися
 i, j: Integer;
 List: TNetwork;
 S: TString;
begin
 Result:='';
 try
  for i:=0 to Count - 1 do begin
   List:=Objects[i] as TNetwork;
   for j:=0 to List.Count - 1 do begin
    S:=List[j];
    CleanUp(S);
    if EqualText(Name, S) then begin
     Result:=Strings[i];
     raise ComputerFound.Create('');
    end;
   end;
  end;
 except
  if not (ExceptObject is ComputerFound) then raise;
 end;
end;

function TNetworkN.FreeRefObj(Index: Integer;
  var Obj: TStringObject): Integer;
begin
 FreeAndNil(Obj.FRefObj);
 Result:=0;
end;
```

```
class function TNetworkN.GetDisplayName(ShellFolder:
                                IShellFolder; PIDL: PItemIDList): TString;
var
// об'ява типів даних що будуть використовуватися
 StrRet: TStrRet;
 P: PChar;
begin
 Result := '';
 ShellFolder.GetDisplayNameOf(PIDL, SHGDN_NORMAL, StrRet);
 case StrRet.uType of
  STRRET_CSTR: SetString(Result, StrRet.cStr, lStrLen(StrRet.cStr));
  STRRET_OFFSET: begin
   P := @PIDL.mkid.abID[StrRet.uOffset - SizeOf(PIDL.mkid.cb)];
   SetString(Result, P, PIDL.mkid.cb - StrRet.uOffset);
  end;
  STRRET_WSTR: Result := StrRet.pOleStr;
 end;
 CleanUp(Result, True);
end;

function TNetworkN.GetPIDLSize(IDList: PItemIDList): Integer;
begin
 Result := 0;
 if Assigned(IDList) then begin
  Result := SizeOf(IDList^.mkid.cb);
  while IDList^.mkid.cb <> 0 do begin
   Result := Result + IDList^.mkid.cb;
   IDList := NextPIDL(IDList);
  end;
 end;
end;

function TNetworkN.GetPrevPIDL(PIDL: PItemIDList): PItemIDList;
var
// об'ява типів даних що будуть використовуватися
 Temp: PItemIDList;
begin
 Temp := CopyPIDL(PIDL);
 if Assigned(Temp) then StripLastID(Temp);
 if Temp.mkid.cb <> 0 then Result:=Temp else Result:=nil;
end;

function TNetworkN.GetWorkgroup(Name: TString): TNetwork;
var
// об'ява типів даних що будуть використовуватися
 Index: Integer;
begin
 Index:=IndexOf(Name);
 if Index<>-1 then Result:=Objects[Index] as TNetwork else Result:=nil;
end;

procedure TNetworkN.Insert(Index: Integer; const S: string);
begin
end;

procedure TNetworkN.ListComputers(Strings: TStrings);
var
// об'ява типів даних що будуть використовуватися
 i, j: integer;
```

```
 L: TNetwork;
 S: TString;
begin
 Strings.BeginUpdate;
 try
  Strings.Clear;
  for i:=0 to Count - 1 do begin
   L:=Objects[i] as TNetwork;
   for j:=0 to L.Count - 1 do begin
    S:=L[j];
    CleanUp(S);
    Strings.Add(S);
   end;
  end;
 finally
  Strings.EndUpdate;
 end;
end;

procedure TNetworkN.ListNetwork(Strings: TStrings);
var
// об'ява типів даних що будуть використовуватися
 List: TStringList;
 i: Integer;
begin
 List:=TStringList.Create;
 try
  List.AddStrings(Self);
  for i:=0 to List.Count - 1 do List.Objects[i]:=TObject(1);
  for i:=0 to Count - 1 do begin
   List.AddStrings(Objects[i] as TStrings);
  end;
  for i:=Count to List.Count - 1 do List.Objects[i]:=nil;
  List.Sort;
  Strings.Assign(List);
 finally
  List.Free;
 end;
end;

function TNetworkN.NextPIDL(IDList: PItemIDList): PItemIDList;
begin
 Result := IDList;
 Inc(PChar(Result), IDList^.mkid.cb);
end;

 function TNetworkN.OriginFolder: IShellFolder;
 var  // об'ява змінних
  Desktop: IShellFolder;
  S: TString;
  P: PWideChar;
  Len, Flags: LongWord;
  Machine, Workgroup, Network: PItemIDList;
 begin
  S:='\\'+GetComputerName;
  Len:=Length(S);
  P:=StringToOleStr(S);
  Flags:=0;
  SHGetDesktopFolder(Desktop);
```

```
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
 Workgroup:=GetPrevPIDL(Machine);
 try
   Network:=GetPrevPIDL(Workgroup);
  try
   Desktop.BindToObject(Network, nil, IShellFolder, Pointer(Result));
  finally
   DisposePIDL(Network);
  end;
 finally
  DisposePIDL(Workgroup);
 end;
end;

function TNetworkN.OriginFolderNT: IShellFolder;
var
// об'ява типів даних що будуть використовуватися
 Desktop: IShellFolder;
 S: TString; W: WideString; P: PWideChar;
 Len, Flags: LongWord;
 Machine, Workgroup, Network: PItemIDList;
 NetShell: IShellFolder;
 Enum: IEnumIDList;
 ID: PItemIDList;
begin
 S:='\\'+GetComputerName;
 Len:=Length(S);
 W:=S; P:=PWideChar(W);
 SHGetDesktopFolder(Desktop);
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
 Workgroup:=GetPrevPIDL(Machine);
 Network:=GetPrevPIDL(Workgroup);
 Desktop.BindToObject(Network, nil, IShellFolder, NetShell);
 Enum:=EnumObjects(NetShell);
 Enum.Next(1, ID, Flags);
 NetShell.BindToObject(ID, nil, IShellFolder, Pointer(Result));
 DisposePIDL(Network);
 DisposePIDL(Workgroup);
end;

class procedure TNetworkN.ParseFolder(Folder: IShellFolder;
  Items: TStringObjectList; StorePIDLs: Boolean);
var
// об'ява типів даних що будуть використовуватися
 ID: PItemiDList;
 EnumList: IEnumIDList;
 NumIDs: LongWord;
 S: TString;
 Index: Integer;
begin
 Items.BeginUpdate;
 try
  Items.Clear;
  EnumList:=EnumObjects(Folder);
  if Assigned(EnumList) then while EnumList.Next(1, ID, NumIDs) = S_OK do begin
   S:=GetDisplayName(Folder, ID);
   Index:=Items.Add(S);
   if StorePIDLs then Items.Data[Index]:=ID;
  end;
```

```
 finally
  Items.EndUpdate;
 end;
end;

class procedure TNetworkN.ParseFolderEx(Folder: IShellFolder;
  Items: TStrings);
var
// об’ява типів даних що будуть використовуватися
 ID: PItemiDList;
 EnumList: IEnumIDList;
 NumIDs: LongWord;
 S: TString;
begin
 Items.BeginUpdate;
 try
  Items.Clear;
  EnumList:=EnumObjects(Folder);
  if Assigned(EnumList) then while EnumList.Next(1, ID, NumIDs) = S_OK do begin
   S:=GetDisplayName(Folder, ID);
   Items.Add(S);
  end;
 finally
  Items.EndUpdate;
 end;
end;

procedure TNetworkN.Refresh;
var
// об’ява типів даних що будуть використовуватися
 Network: IShellFolder;
 Workgroup: IShellFolder;
 i: Integer;
begin
 try
  if WinNT and (not Win2K) then Network:=OriginFolderNT else
   Network:=OriginFolder;
  ParseFolder(Network, Self, True);
  for i:=0 to Count - 1 do begin
   Network.BindToObject(PItemIDList(Data[i]), nil, IShellFolder, Workgroup);
   ParseFolder(Workgroup, Objects[i] as TStringObjectList, False);
   Workgroup:=nil;
  end;
 except
  raise ECannotFindNetwork.Create(SCannotFindNetwork);
 end;
end;

procedure TNetworkN.StripLastID(IDList: PItemIDList);
var
// об’ява типів даних що будуть використовуватися
 MarkerID: PItemIDList;
begin
 MarkerID := IDList;
 if Assigned(IDList) then begin
  while IDList.mkid.cb <> 0 do begin
   MarkerID := IDList;
   IDList := NextPIDL(IDList);
  end;
```

```
  MarkerID.mkid.cb := 0;
 end;
end;

procedure GetIPAddresses(Network: TNetworkNeighborhood; List: TStrings);
var
// об'ява типів даних що будуть використовуватися
 Error: DWORD;
 HostEntry: PHostEnt;
 Data: WSAData;
 Address: In_Addr;
 i: Integer;
 TmpList: TStringList;
 S: TString;
begin
List.BeginUpdate;
 try
  List.Clear;
  Error:=WSAStartup(MakeWord(1, 1), Data);
  if Error = 0 then begin
   TmpList:=TStringList.Create;
   try
    Network.ListComputers(TmpList);
    for i:=0 to TmpList.Count - 1 do begin
     HostEntry:=gethostbyname(PChar(TmpList[i]));
     Error:=GetLastError;
     if Error <> 0 then S:='Unknown' else begin
      Address:=PInAddr(HostEntry^.h_addr_list^)^;
      S:=inet_ntoa(Address);
     end;
     List.Add(Format('%s [%s]', [TmpList[i], S]));
    end;
   finally
    TmpList.Free;
   end;
  end else begin
   List.Add('Error');
  end;
 finally
  List.EndUpdate;
 end;
end;

function GetShellFolder(ComputerName: TString): IShellFolder;
var  // об'ява змінних
 S: TString;
 W: WideString;
 P: PWideChar;
 Desktop: IShellFolder;
 Len, Flags: LongWord;
 Machine: PItemIDList;
begin
 S:=ComputerName;
 if Pos('\\', S) <> 1 then S:='\\'+S;
 Len:=Length(S);
 W:=S;
 P:=@W[1];
 SHGetDesktopFolder(Desktop);
 Desktop.ParseDisplayName(0, nil, P, Len, Machine, Flags);
```

```
 Desktop.BindToObject(Machine, nil, IShellFolder, Pointer(Result));
end;

function EnumSharedResources(ComputerName: TString; List: TStrings): Boolean;
var
// об’ява типів даних що будуть використовуватися
 ShellFolder: IShellFolder;
begin
 ShellFolder:=GetShellFolder(ComputerName);
 Result:=Assigned(ShellFolder);
 if Result then TNetworkN.ParseFolderEx(ShellFolder, List);
end;

function GetIPAddress(NetworkName: TString): TString;
var
// об’ява типів даних що будуть використовуватися
 Error: DWORD;
 HostEntry: PHostEnt;
 Data: WSAData;
 Address: In_Addr;
begin
 Error:=WSAStartup(MakeWord(1, 1), Data);
 if Error = 0 then begin
  HostEntry:=gethostbyname(PChar(NetworkName));
  Error:=GetLastError();
  if Error = 0 then begin
   Address:=PInAddr(HostEntry^.h_addr_list^)^;
   Result:=inet_ntoa(Address);
  end else begin
   Result:='Unknown';
  end;
 end else begin
  Result:='Error';
 end;
 WSACleanup();
end;
end.
```

Модуль програми A3.pas

```
unit A3;
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Вихідний код до бакалаврської роботи
на тему: Програмне забезпечення системи Network Performance Monitoring
Виконав: студент  4 курсу Хуторний Дмитро Олексійович
Керівник: Дреєв О.М
2021 рік
}
Interface
// опис/об'ява
// Додавання бібліотек що будуть використовуватись
uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls, StdCtrls,
   Buttons, ExtCtrls;

// Опис типів даних та класів
type
  TAboutBox = class(TForm) // розроблений клас
    Panel1: TPanel;
    OKButton: TBitBtn;
    ProgramIcon: TImage;
    ProductName: TLabel;
    Version: TLabel;
    Copyright: TLabel;
    Label1: TLabel;
    // Розроблена функція ПЗ
     procedure OKButtonClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end; //кінець
  var
// об'ява типів даних що будуть використовуватися
  AboutBox: TAboutBox;
Implementation // модуль
{$R *.DFM} //ресурси модуля
// Розроблена функція ПЗ
 procedure TAboutBox.OKButtonClick(Sender: TObject);
begin //початок
 close;
end; //кінець
end.
```

Модуль програми A4.pas

```
unit A4; // модуль

interface
// опис-об'ява

uses // Додавання бібліотек що будуть використовуватись
   Windows, // Вікна VCL
   WInSock, // Сокети
   Snmp, // робота з Інтернетом
   SysUtils, // системні функції
   Classes; // Класи VCL

const
   mibLen = $0A;

Type
 // Опис типів даних та класів

   TMibId = array[1..mibLen] of integer;

   TIpConnInfo = class
// новий клас
   private
      FRemoteIp :TInAddr;
      FRemotePort :integer;
      FState :DWORD;
      FLocalIp :TInAddr;
      FLocalPort :integer;
      FProto :ShortString;
      function GetLocalPort :Integer;
      function GetLocalIpString :String;
      function Ip2Str( const Ip :TInAddr ):String;
      function GetRemotePort :Integer;
      function GetRemoteIpString :String;
      function GetStateString :String;
   public
      constructor Create( const aProto:String );
      property IpProtoName :ShortString
         read FProto;
      property LocalIp :TInAddr
         read FLocalIp;
      property LocalPort :Integer
         read GetLocalPort;
      property LocalIpString :String
         read GetLocalIpString;
      property RemoteIpString :String
         read GetRemoteIpString;
      property RemoteIp :TInAddr
         read FRemoteIp;
      property RemotePort :Integer
         read GetRemotePort;
      property State :DWORD
         read FState;
      property StateString :String
         read GetStateString;
   end;
```

```
   EConnListError = class(Exception);
      EConnListLockError = class(EConnListError);
      EConnListUnlockError = class(EConnListError);
      EConnListRefreshError = class(EConnListError);

   TIpConnListStatus = ( connlist_ready, connlist_refreshing );

   TIpProtocol = (udp_ip, tcp_ip);
   TIpProtocols = set of TIpProtocol;

   TFnugryIpConnectionList = class(TComponent)
   private
      FConnections :TList;
      FWsInited :Bool;
      FSnmpLib :THandle;
      FSnmpQueryProc :Pointer;
      FSnmpInitProc :Pointer;
      FProtocols :TIpProtocols;
      FStatus :TIpConnListStatus;
      FOnStatusChange :TNotifyEvent;
      FAccessMutex :THandle;
      FPollForTrapEvent :THandle;
      FSupportedViewRoot :TAsnObjectIdentifier;
      function GetConnCount :Integer;
      function GetConnections( Index :Integer ):TIpConnInfo;
   protected
      procedure StatusChange; virtual;
      procedure SetStatus( Value :TIpConnListStatus ); virtual;
      procedure DoLock;
      procedure DoUnlock;
      procedure Clear;
   public
      constructor Create( aOwner :TComponent ); override;
      destructor Destroy; override;
      procedure Refresh;
      function Lock( Timeout :DWORD ):Bool;
      function Unlock :Bool;
      property Status :TIpConnListStatus
      read FStatus;
      property Connections[ Index :Integer ] :TIpConnInfo
      read GetConnections; default;
      property ConnCount :Integer
      read GetConnCount;
   published
      property Protocols :TIpProtocols
      read FProtocols write FProtocols;
      property OnStatusChange :TNotifyEvent
      read FOnStatusChange write FOnStatusChange;
   end;

   ENetstatError = class(Exception);

   TNetstatCounterObject = class
// новий клас
   private
      FName :String;
      FId   :TMibId;
      FDescription :String;
      FWsInited :Bool;
```

```
      FSnmpLib :THandle;
      FSnmpQueryProc :Pointer;
      FSnmpInitProc :Pointer;
      function GetValue :DWORD;
   public
constructor Create( const aName, aDesc :String; const aId :TMibId );
      destructor Destroy; override;
      property Description :String
         read FDescription;
      property Name :String
         read FName;
      property Id :TMibId
         read FId;
      property Value :DWORD
         read GetValue;
   end;

   TCounterList = class(TComponent) // новий клас
   private
      FCounters :TStringList;
      function GetCount :Integer;
      function GetCounters( Index :Integer ):TNetstatCounterObject;
   public
      constructor Create( aOwner :TComponent ); override;
      destructor Destroy; override;
      procedure Clear;
      procedure AddCounter( aCounter :TNetstatCounterObject );
      function IndexOf( const aName :String ):Integer;
      property Count :Integer
         read GetCount;
      property Counters[Index :Integer] :TNetstatCounterObject
         read GetCounters; default;
   end;
   TIpStats = class(TCounterList) // новий клас
   public
      constructor Create( aOwner :TComponent ); override;
   end;
   TIcmpStats = class(TCounterList) // новий клас
   public
      constructor Create( aOwner :TComponent ); override;
   end;
   TTCPStats = class(TCounterList) // новий клас
   public
      constructor Create( aOwner :TComponent ); override;
   end;
   TUdpStats = class(TCounterList) // новий клас
   public
      constructor Create( aOwner :TComponent ); override;
   end;
implementation // модуль

const // об’ява констант
  DEFAULT_LOCK_TIMEOUT  = 100;
  SNMP_LIB_NAME         = 'inetmib1.dll'; // використовуваєма бібліотека
  SNMP_INITPROC_NAME    = 'SnmpExtensionInit'; // Початкова функція
  SNMP_QUERYPROC_NAME   = 'SnmpExtensionQuery'; // Функція запиту

Type
 // Опис типів даних та класів
```

```
  TSnmpInitProc = function(
     dwTimeZeroReference : DWORD;
     Var hPollForTrapEvent : THandle;
     Var SupportedView : TAsnObjectIdentifier) : BOOL; stdcall;

  TSnmpQueryProc = function(
     RequestType : Byte;
     Var VariableVindings : TRFC1157VarBindList;
     Var ErrorStatus : TAsnInteger;
     Var ErrorIndex : TAsnInteger) : BOOL; stdcall;

// Реалізація методу класу
{ TIpStats }

constructor TIpStats.Create( aOwner :TComponent );
begin
   inherited;
   AddCounter( TNetStat.Create( 'ipDefaultTTL', '', mib_ipDefaultTTL));
   AddCounter( TNetStat.Create( 'ipInReceives', '', mib_ipInReceives));
   AddCounter( TNetStat.Create( 'ipInHdrErrors', '', mib_ipInHdrErrors));
end;

// Реалізація методу класу
{ TIcmpStats }
// Реалізація методу класу
{ TNetstatCounterObject }
function TNetStat.GetValue :DWORD;
var
// об'ява типів даних що будуть використовуватися
   varBind     :TRFC1157VarBind;
   varBindList :TRFC1157VarBindList;
   errorStatus :TAsnInteger;
   errorIndex  :TAsnInteger;
begin
   varBindList.List := @varBind;
   varBindList.len := 1;
   fillchar(varBind, SizeOf(varBind), 0);
   varBind.Name.idLength := mibLen;
   varBind.Name.ids := @FId;
   if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
      varBindList, errorStatus, errorIndex) then
      raise ENetstatError.Create(m_err_query);
   if not (varBindList.list.value.asnType in
    [ASN_GAUGE32, ASN_INTEGER, ASN_INTEGER32, ASN_COUNTER32, ASN_UNSIGNED32])
then
     raise ENetstatError.Create(m_err_invtype);
    result := varBindList.list.value.Counter;
end;

constructor TNetStat.Create(
  const aName, aDesc :String; const aId :TMibId );
var
   WSData :TWSAData;
   PollForTrapEvent :THandle;
   SupportedViewRoot :TAsnObjectIdentifier;
begin
   inherited Create;
   FWsInited := WsaStartup($0101, WSData ) = 0;
```

```
   if not FWsInited then
      raise EConnListError.Create(m_err_wsstartup);
   FSnmpLib := LoadLibrary(SNMP_LIB_NAME);
   if FSnmpLib = 0 then
      raise EConnListError.Create(m_err_loadlib);
   FSnmpInitProc := GetProcAddress(FSNmpLib, SNMP_INITPROC_NAME);
   FSnmpQueryProc := GetProcAddress(FSNmpLib, SNMP_QUERYPROC_NAME);
   if ( ( FSnmpQueryProc = Nil ) or ( FSnmpInitProc = Nil ) ) then
      raise EConnListError.Create(m_err_loadlib);
   if not TSnmpInitProc(FSnmpInitProc)(GetTickCount,
      PollForTrapEvent, SupportedViewRoot) then
      raise EConnListError.Create(m_err_initlib);
   FName := aname;
   FDescription := aDesc;
   FId := aid;
   dec(Fid[8]);
end;

destructor TNetStat.Destroy;
begin
   if FSnmpLib <> 0 then FreeLibrary(FSnmpLib);
   if FWsInited then WSACleanup;
   inherited;
end;

// Реалізація методу класу
{ TCounterList }

procedure TCounterList.AddCounter( aCounter :TNetstatCounterObject );
begin
   assert( assigned( aCounter ));
   FCounters.AddObject( aCounter.Name, aCounter );
end;

function TCounterList.GetCount :Integer;
begin
   assert( assigned( FCounters ));
   result := FCounters.Count;
end;

function TCounterList.GetCounters(Index:Integer ):TNetstatCounterObject;
begin
   assert( assigned( FCounters ));
   result := TNetstatCounterObject(FCounters.Objects[Index]);
end;

function TCounterList.IndexOf( const aName :String ):Integer;
begin
   result := FCounters.IndexOf( aName );
end;

constructor TCounterList.Create( aOwner :TComponent );
begin
   inherited;
   FCounters := TStringList.Create;
end;
```

```
destructor TCounterList.Destroy;
begin
   Clear;
   if assigned(FCounters) then FCounters.Free;
   inherited;
end;

procedure TCounterList.Clear;
var
// об’ява типів даних що будуть використовуватися
   i :integer;
   c :TObject;
begin
   if assigned(FCounters) then
      for i := FCounters.Count-1 downto 0 do
         begin
            c := FCounters.Objects[i];
            FCounters.Delete(i);
            if assigned(c) then c.free;
         end;
end;
// Реалізація методу класу
{ TIpConNInfo }

constructor TIpConNInfo.Create( const aProto :String );
begin
   inherited Create;
   FState := LISTEN;
   FProto := aProto;
end;

function TIpConNInfo.Ip2Str( const Ip :TInAddr ):String;
begin
   result := format('%d.%d.%d.%d',
     [Integer(ip.s_un_b.s_b1),
      Integer(ip.s_un_b.s_b2),
      Integer(ip.s_un_b.s_b3),
      Integer(ip.s_un_b.s_b4)]);
end;

function TIpConNInfo.GetLocalPort :Integer;
begin
   result := FLocalPort;
end;

function TIpConNInfo.GetLocalIpString :String;
begin
   result := Ip2Str(FLocalIp);
end;

function TIpConNInfo.GetRemotePort :Integer;
begin
   if State = LISTEN then
      result := 0
   else
      result := FRemotePort;
end;
```

```
function TIpConNInfo.GetRemoteIpString :String;
begin
   result := Ip2Str(FRemoteIp);
end;

function TIpConNInfo.GetStateString :String;
const
   state_name :array[CLOSED..TCB_DISCARD] of string[16] =
   ('CLOSED',
    'LISTEN',
    'SYN_SENT',
    'SYN_RECEIVED',
    'ESTABLISHED',
    'CLOSE_WAIT',
    'FIN_WAIT_1',
    'CLOSING',
    'LAST_ACK',
    'FIN_WAIT_2',
    'TIME_WAIT',
    'TCB_DISCARD'
   );
begin
   if State in [CLOSED..TCB_DISCARD] then
      result := state_name[state]
   else
      result := 'UNKNOWN';
end;

// Реалізація методу класу
{ TFnugryIpConnectionList }

function TFnugryIpConnectionList.GetConnCount :Integer;
begin
   assert(assigned(FConnections));
   result := FConnections.Count;
end;

function TFnugryIpConnectionList.GetConnections( Index :Integer ):TIpConnInfo;
begin
   assert(assigned(FConnections));
   result := FConnections[Index];
end;

procedure TFnugryIpConnectionList.Clear;
var
// об'ява типів даних що будуть використовуватися
   i :integer;
   p :TIpConnInfo;
begin
   if assigned(FConnections) then
      for i := FConnections.Count-1 downto 0 do
         begin
            p := FConnections[i];
            assert(assigned(p));
            FConnections.Delete(i);
            p.free;
         end;
end;
```

```
procedure TFnugryIpConnectionList.StatusChange;
begin
   if assigned(FOnStatusChange) then FOnStatusChange(Self);
end;

procedure TFnugryIpConnectionList.SetStatus( Value :TIpConnListStatus );
begin
   if FStatus <> Value then
      begin
        FStatus := Value;
        StatusChange;
      end;
end;

procedure TFnugryIpConnectionList.DoLock;
begin
   if not Lock(DEFAULT_LOCK_TIMEOUT) then
      raise ECOnnListLockError.Create(m_err_lock);
end;

procedure TFnugryIpConnectionList.DoUnlock;
begin
   if not UnLock then
      raise ECOnnListUnLockError.Create(m_err_unlock);
end;

constructor TFnugryIpConnectionList.Create( aOwner :TComponent );
var
// об'ява типів даних що будуть використовуватися
   WSData :TWSAData;
begin
   inherited;
   FProtocols := [udp_ip, tcp_ip];
   FWsInited := WsaStartup($0101, WSData ) = 0;
   if not FWsInited then
      raise EConnListError.Create(m_err_wsstartup);
   FSnmpLib := LoadLibrary(SNMP_LIB_NAME);
   if FSnmpLib = 0 then
      raise EConnListError.Create(m_err_loadlib);
   FSnmpInitProc := GetProcAddress(FSNmpLib, SNMP_INITPROC_NAME);
   FSnmpQueryProc := GetProcAddress(FSNmpLib, SNMP_QUERYPROC_NAME);
   if ( ( FSnmpQueryProc = Nil ) or ( FSnmpInitProc = Nil ) ) then
      raise EConnListError.Create(m_err_loadlib);
   if not TSnmpInitProc(FSnmpInitProc)(GetTickCount,
      FPollForTrapEvent, FSupportedViewRoot) then
      raise EConnListError.Create(m_err_initlib);
   FAccessMutex := CreateMutex(nil, false, nil);
   if FAccessMutex = 0 then
      raise ECOnnListError.Create(m_err_alloc);
   FCOnnections := TList.Create;
   FStatus := connlist_ready;
end;

destructor TFnugryIpConnectionList.Destroy;
begin
   Clear;
   if assigned(FConnections) then FCOnnections.Free;
   if FAccessMutex <> 0 then CloseHandle(FAccessMutex);
   if FSnmpLib <> 0 then FreeLibrary(FSnmpLib);
```

```
   if FWsInited then WSACleanup;
   inherited;
end;

function TFnugryIpConnectionList.Lock( Timeout :DWORD ):Bool;
begin
   result := WaitForSingleObject(FAccessMutex, Timeout ) = WAIT_OBJECT_0;
end;

function TFnugryIpConnectionList.Unlock :Bool;
begin
  result := ReleaseMutex(FAccessMutex);
end;

procedure TFnugryIpConnectionList.Refresh;

   procedure ReadTcpTable;
   var
// об'ява типів даних що будуть використовуватися
      varBind     :TRFC1157VarBind;
      varBindList :TRFC1157VarBindList;
      errorStatus :TAsnInteger;
      errorIndex  :TAsnInteger;
      ConnIndex   :Integer;
      Info        :TIpConnInfo;
      ListTail    :Integer;
   begin
      fillchar( varBindList, SizeOf(varBindList), 0);
      varBindList.List := @varBind;
      varBindList.len := 1;
      fillchar(varBind, SizeOf(varBind), 0);
      varBind.Name.idLength := mibLen;
      varBind.name.ids := @mib_tcpConnTable;
      ListTail    := FConnections.Count;
      if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
         varBindList, errorStatus, errorIndex) then
         raise EConnListRefreshError.Create(m_err_query);
      if varBindList.list.value.asnType = ASN_NULL then
         exit;
      while (varBindList.list.value.asnType = ASN_INTEGER) do
      begin
         Info := TIpConnInfo.Create('TCP');
         Info.FState := varBindList.list.value.Counter;
         FCOnnections.Add(Info);
if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
            varBindList, errorStatus, errorIndex) then
            raise EConnListRefreshError.Create(m_err_query);
      end;
      if varBindList.list.value.asnType = ASN_NULL then
         raise EConnListRefreshError.Create(m_err_queryend);
      ConnIndex := ListTail;
      while (varBindList.list.value.asnType = ASN_RFC1155_IPADDRESS) do
      begin
         move(varBindList.list.value.address.stream^,
            Connections[ConnIndex].FLocalIP, sizeof(TInAddr));
         if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
            varBindList, errorStatus, errorIndex) then
            raise EConnListRefreshError.Create(m_err_query);
         inc(ConnIndex);
```

```
      end;
      if varBindList.list.value.asnType = ASN_NULL then
         raise EConnListRefreshError.Create(m_err_queryend);
      { читання локального порту }
      ConnIndex := ListTail;
      while (varBindList.list.value.asnType = ASN_INTEGER) do
      begin
         Connections[ConnIndex].FLocalPort := varBindList.list.value.counter;
         if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
            varBindList, errorStatus, errorIndex) then
            raise EConnListRefreshError.Create(m_err_query);
         inc(ConnIndex);
      end;
      if varBindList.list.value.asnType = ASN_NULL then
         raise EConnListRefreshError.Create(m_err_queryend);
      ConnIndex := ListTail;
      while (varBindList.list.value.asnType = ASN_RFC1155_IPADDRESS) do
      begin
         move(varBindList.list.value.address.stream^,
            Connections[ConnIndex].FRemoteIP,
            sizeof(TInAddr));
         if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
            varBindList, errorStatus, errorIndex) then
            raise EConnListRefreshError.Create(m_err_query);
         inc(ConnIndex);
      end;
      if varBindList.list.value.asnType = ASN_NULL then
         raise EConnListRefreshError.Create(m_err_queryend);
      { читання локального порту }
      ConnIndex := ListTail;
      while (varBindList.list.value.asnType = ASN_INTEGER) do
      begin
         Connections[ConnIndex].FRemotePort := varBindList.list.value.counter;
         if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
            varBindList, errorStatus, errorIndex) then
            raise EConnListRefreshError.Create(m_err_query);
         inc(ConnIndex);
      end;
   end;

   procedure ReadUdpTable;
    var
// об'ява типів даних що будуть використовуватися
      varBind     :TRFC1157VarBind;
      varBindList :TRFC1157VarBindList;
      errorStatus :TAsnInteger;
      errorIndex  :TAsnInteger;
      ConnIndex   :Integer;
      Info        :TIpConnInfo;
      ListTail    :Integer;
    begin
      fillchar( varBindList, SizeOf(varBindList), 0);
      varBindList.List := @varBind;
      varBindList.len := 1;
      fillchar(varBind, SizeOf(varBind), 0);
      varBind.Name.idLength := mibLen;
      varBind.name.ids := @mib_udpTable;
      ListTail := FConnections.Count;
      if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
```

```
            varBindList, errorStatus, errorIndex) then
            raise EConnListRefreshError.Create(m_err_query);
         if varBindList.list.value.asnType = ASN_NULL then
            exit;
{ читання локального порту }
         while (varBindList.list.value.asnType = ASN_RFC1155_IPADDRESS) do
         begin
            Info := TIpConnInfo.Create('UDP');
            move(varBindList.list.value.address.stream^,
               Info.FLocalIP,
               sizeof(TInAddr));
            FConnections.Add(Info);
            if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
               varBindList, errorStatus, errorIndex) then
               raise EConnListRefreshError.Create(m_err_query);
         end;
         if varBindList.list.value.asnType = ASN_NULL then
            raise EConnListRefreshError.Create(m_err_queryend);
         { читання локального порту }
         ConnIndex := ListTail;
         while (varBindList.list.value.asnType = ASN_INTEGER) do
         begin
Connections[ConnIndex].FLocalPort := varBindList.list.value.counter;
if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
               varBindList, errorStatus, errorIndex) then
               raise EConnListRefreshError.Create(m_err_query);
            inc(ConnIndex);
         end;
     end;

begin
   DoLock;
   try
      if FStatus <> connlist_ready then
         raise EConnListError.Create(m_err_inv_state);
      SetStatus( connlist_refreshing );
      Clear;
      if tcp_ip in FProtocols then
         ReadTcpTable;
      if udp_ip in FProtocols then
         ReadUdpTable;
   finally
      DoUnlock;
      SetStatus(connlist_ready);
   end;
end;
end.
```

НАЛАШТУВАННЯ КОМПІЛЯТОРА ДЛЯ КОМПІЛЯЦІЇ РОЗРОБЛЕНОГО ПЗ

```
[Compiler]
A=1 B=0 C=1 D=1
E=0 F=0 G=1 H=1
I=1 J=1 K=0 L=1
M=0 N=1 O=1 P=1
Q=0 R=0 S=0 T=0
U=0 V=1 W=0 X=1
Y=0 Z=1
ShowHints=1
ShowWarnings=1
UnitAliases=WinTypes=Windows;WinProcs=Windows;DbiTypes=BDE;
            DbiProcs=BDE;DbiErrs=BDE;
[Linker]
MapFile=0
OutputObjs=0
ConsoleApp=1
DebugInfo=0
MinStackSize=16384
MaxStackSize=1048576
ImageBase=4194304
ExeDescription=
[Directories]
OutputDir=
UnitOutputDir=
SearchPath=
Packages=VCLX30;VCL30;VCLDB30;VCLDBX30;
         INETDB30;INET30;VCLSMP30;QRPT30;
         TEEUI;
Conditionals=
DebugSourceDirs=
UsePackages=0
[Parameters]
RunParams=
HostApplication=
[Version Info]
IncludeVerInfo=0
AutoIncBuild=0
MajorVer=1
MinorVer=0
Release=0
Build=0
Debug=0
PreRelease=0
Special=0
Private=0
DLL=0
Locale=3082
CodePage=1252
[Version Info Keys]
CompanyName=
FileDescription=
FileVersion=1.0.0.0
ProductName=2021 year
ProductVersion=1.0.0.0
Comments= None
```