Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності”

Виконав здобувач вищої освіти
IV курсу, групи КБ-21-3СК
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Плужник В.О.
«____» _____ 2024 р.

Керівник проекту
кандидат технічних наук
_____ Смірнова Т.В.
«____» _____ 2024 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2024 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Плужнику Владиславу Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності*

2. Керівник роботи *Смірнова Тетяна Віталіївна, канд. техн. наук*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 136-02 від 01.04.2024 року

3. Строк подання студентом роботи до захисту *23.05.2024 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності*

5. 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 кібербезпеки в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

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

7. Дата видачі завдання « 17 » січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2024 р.	
3.	Розробка моделі компонента	20.03.2024 р.	
4.	Розробка структур даних	25.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2024 р.	
6.	Програмування алгоритмів	10.04.2024 р.	
7.	Оформлення ПЗ	17.04.2024 р.	
8.	Попередній захист роботи	23.05.2024 р.	

Дата видачі завдання
« 17 » січня 2024 р.

Підпис керівника
_____ Смірнова Т.В.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2024 р.

Підпис здобувача
_____ Плужник В.О.
(прізвище та ініціали)

АНОТАЦІЯ

Плужник В.О. Програмне забезпечення системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

Метою розробки є програмне забезпечення системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

Результат роботи – програмна реалізація системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

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 з ОС Windows 10/11.

Програму розроблено в середовищі RAD Studio Delphi.

Ключові слова: кібербезпека, резервне копіювання налаштувань

ABSTRACT

Pluzhnyk V.O. Cybersecurity system software backs up LAN router settings to ensure availability. 125 Cyber security. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this final qualification work at the first (bachelor) level of higher education, software is developed, which is intended for a cyber security system for backing up the settings of local network routers to ensure availability.

The goal of the development is the cyber security system software to backup the settings of the LAN routers to ensure availability.

The result of the work is a software implementation of a cyber security system for backing up local network router settings to ensure availability.

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.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on a PC with Windows 10/11 OS.

The program was developed in the RAD Studio Delphi environment.

Keywords: cyber security, backup of settings

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер;

TCP/IP – Transport Control Protocol/Internet Protocol – протокол роботи мережі інтернет;

PPTP – Point-to-Point-Tunneling Protocol – протокол створення захищеного каналу при доступі віддалених користувачів через публічні мережі;

ПК – персональний комп’ютер;

ЦО – цивільна оборона;

ОС – операційна система;

ПЕОМ – персональна електронно обчислювальна машина;

КС – комп’ютерна система;

ГМ – глобальні мережі;

ЛВМ – локальна віртуальна мережа;

ООП – об'єктно-орієнтоване програмування;

PLL – Цикл Блокування Стадії;

URL – universal resource locator – локатор ресурсів інтернет;

HTML – мова розмітки гіпертекстових документів.

ВСТУП

Актуальність теми. На сьогоднішній день резервне копіювання даних стало невід'ємною частиною щоденних завдань, виконуваних системними адміністраторами. І це не дивно, так як збитки, які компанія може понести у випадку втрати даних або змушеного простою, найчастіше набагато більше вартості встаткування, використовуваного для зберігання даних. Існує маса різних програмних продуктів, що дозволяють здійснювати автоматичне резервне копіювання даних практично в будь-якому форматі, від «знімка» стану операційної системи до бекапу баз даних. Однак все різноманіття застосовне лише до програмних рішень. З апаратними рішеннями, зокрема, з активним мережним устаткуванням, справа небагато складніша. Звичайно, багато хто можуть заперечити, що енергонезалежна пам'ять маршрутизаторів і комутаторів більше стійка до збоїв в енергомережі й що цілком достатньо, скажемо, раз на місяць робити вручну копію робочої конфігурації свого встаткування й зберігати її на TFTP-сервер [1].

Однак у випадку конфігурації мережі, що динамічно змінюється, такий підхід буде, м'яко говорячи, не занадто гарний. Наприклад, на маршрутизаторах телекомунікаційної компанії, де налаштування інтерфейсів, статичні маршрути або access-list можуть мінятися щодня. Здійснювати резервне копіювання вручну в такій мережі дуже складно, особливо якщо кількість маршрутизаторів і комутаторів більше п'яти. У такій ситуації нам необхідно здійснювати щоденне, автоматичне резервне копіювання робочих конфігурацій активного мережного встаткування.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

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

– Огляд існуючих систем резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

– Дослідження системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

– Програмна реалізація системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для забезпечення кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності. Конфігурація маршрутизатора відноситься до числа критично важливих даних, тому обов'язково повинне здійсняться її регулярне резервне копіювання. Тоді при виході з ладу маршрутизатора можна буде швидко відновити роботу, замінивши його й скопіювавши збережений конфігураційний файл. Причому зберігати потрібно не тільки останню версію, але й декілька попередніх – якщо через помилку при налаштуванні виникнуть проблеми, це дозволить легко повернуться до попередньої версії. Зміни в конфігурації висилаються адміністраторові поштою. Це особливо зручно, коли одним маршрутизатором управляє кілька людей, оскільки дозволяє бути в курсі змін, зроблених іншими. Для рішення поставленого завдання ми скористаємося розробленим програмним забезпеченням, що буде запускатися за розкладом, віддалено підключатися до кожного маршрутизатора або комутатора за протоколом Telnet або SSH і копіювати робочу конфігурацію на TFTP-сервер.

1.2 Область застосування

Областю застосування системи є локальна мережа. Локальна комп'ютерна мережа – це система, що дозволяє проводити обмін інформацією між пристроями, підключеними до системи. Вона включає в себе програмне забезпечення та апаратну частину, необхідну для підключення пристроїв до комп'ютерних каналів, які взаємодіють між собою.

Базова комунікативна модель складається з:

– джерела (персональний комп'ютер);

– приймача (сервер);

– сполучення між ними (кабель або телефонна лінія).

Основні функції локальної мережі:

– надання користувачам загального доступу до папок, файлів і інших ресурсів;

– спільне використання файлів;

– обмеження, при бажанні доступу до спільних папок і каталогів;

– настройка прав доступу;

– архівація потрібної інформації та зберігання даних на файловому сервері.

– можливість надійно зашифровувати свої дані на файловому сервері, що дає високий рівень захисту від зовнішніх атак.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Для рішення завдання резервного копіювання конфігурації маршрутизатора існує як мінімум дві програми – ciscoconf (порт FreeBSD /usr/ports/net-mgmt/ciscoconf) і rancid. Перша (ciscoconf) копіює конфігурації конфігурацію по RSH і зберігає в RCS. Її перевага – вона це може робити відразу після завершення конфігурації відслідковуючи появу в лозі рядок утримуючий % SYS-5-CONFIG_I. Друга (rancid) більше універсальна й дозволяє працювати не тільки з устаткуванням Cisco. Для своєї роботи вона вимагає збереження в конфігураційному файлі пароля на доступ по telnet і пароль enable. Я вважаю, що це робити небажано з міркувань безпеки. Крім цього існує ще один проект – Pancho. Там так само конфігураційні файли копіюються використовуючи snmp+tftp.

Автоматичне збереження конфігурації пристроїв Cisco

На початку, потрібно встановити TFTP сервер (так само можна використовувати FTP або інший спосіб, я зберігаю в конфігурації по локальній мережі в окремому менеджмент VLAN – тому використовую TFTP без автентифікації).

Під TFTP-сервер можливо використовувати як Linux – так і Windows сервера, у мене для цих цілей є сервер з ОС Windows 2012. Під нього потрібно скачати TFTP сервер – я для цих цілей використовую безкоштовну tftpd32 service edition, вона встановлюється й піднімається як сервіс у системі. Запускаємо програму, указуємо їй папку, у якій буде збережуться конфігураційний файли,

указуємо який IP вона буде використовувати й перевіряємо доступність TFTP-сервера з маршрутизатора простим копіюванням файлу із внутрішнього flash:

```
RT-01#copy flash: tftp:
Source filename []? 3.txt
Address or name of remote host []? 192.168.10.24
Destination filename [3.txt]?
!!!
11335 bytes copied in 0.044 secs (257614 bytes/sec)
RT-01#
```

У мене у внутрішній пам'яті маршрутизатора лежав файл «3.txt» – і я його успішно скопіював на TFTP-сервер.

Спосіб перший. Створення завдання kron.

1) Створення скрипта-політики для бекапу:

```
Router(config)#kron policy-list (ім'я)
Router( config-kron-policy)#cli copy (звідки копіювати) (куди копіювати)
Router( config-kron-policy)#exit
```

Де наступні параметри:

– cli – визначення EXEC CLI команди в завданні політики.

– policy-list – визначення політики, що буде асоціюватися із завданням в інструкції.

Приклад:

```
RT-01(config)#kron policy-list conf_to_tftp
RT-01(config-kron-policy)#cli copy system:/ running-config
tftp://192.168.10.24/ rt-01.txt
```

2) Створюється інструкція для пристроїв с часом і інтервалом виконання завдання:

```
Router(config)#kron
occurrence (name) at (hh:mm) (day/month/oneshot/reccuring)
Router( config-kron-occurrence) #policy-list (ім'я)
```

Приклад:

```
RT-01(config)#kron occurrence daily at 4:00 recurring
RT-01(config-kron-occurrence) #policy-list conf_to_tftp
```

3) Перевірка конфігурації командою show kron.

```
RT-01#sh kron schedule
Kron Occurrence Schedule
daily inactive, will run again in 0 days 15:04:22 at 4:00 on
```

Спосіб другої. Створення архівування.

Архівування з'явилося в пристроях з версії 12.3 – тому, можливо прийдеться обновляти IOS. У цей момент для багатьох пристроїв уже використовуються IOS від версії 15.x і вище. Відповідно на старих пристроях даний функціонал не підтримується.

Подивимося параметри даної команди:

```
RT-01(config)#archive
RT-01(config-archive)#?
Archive configuration commands:
default Set a command to its defaults
exit Exit from archive configuration mode
log Logging commands
maximum maximum number of backup copies
no Negate a command or set its defaults
path path for backups
rollback Rollback parameters
time-period Period of time in minutes to automatically archive the running-
config
write-memory Enable automatic backup generation during write memory
```

Опишу кожний параметр:

– log – налаштування логування;

– maximum – максимальна кількість резервних копій конфігурації (за замовчуванням 10);

– path – шлях, що вказує де зберігаються резервні копії. При завданні імені файлу можна використовувати змінні $H – ім'я пристрою, і $T – поточний час;

– time-period – період часу через який буде автоматично виконуватися архівування поточної конфігурації (у хв), якщо виставити значення 1440 (24 години), те зберігатися буде щодоби й при збереженні конфігурації пристрою;

– write-memory – включає автоматичну генерацію резервної копії конфігурації, після виконання збереження конфігурації;

– hidekeys – приховувати паролі при архівації (хоча ніхто не скасовував використання secret замість password).

Подивимося можливі шляхи для збереження архівів:

```
RT-01(config-archive)#path?
flash0: Write archive on flash0: file system
flash1: Write archive on flash1: file system
flash: Write archive on flash: file system
ftp: Write archive on ftp: file system
http: Write archive on http: file system
https: Write archive on https: file system
rcp: Write archive on rcp: file system
scp: Write archive on scp: file system
tftp: Write archive on tftp: file system
```

Команда так само дозволяє зберігати конфігурацію в різні місця.

Налаштування збереження на TFTP буде виглядати в такий спосіб:

```
RT-01(config)#archive
log config
logging enable
logging persistent reload
hidekeys
path tftp://192.168.10.24/$ H-H-$T
write-memory
```

Тепер при кожному виконанні команди збереження конфігураційного файлу на пристрої створиться файл на віддаленомк tftp сервері.

Перевіряємо працездатність, зберігаємо конфігурацію:

```
RT-01#wr
Building configuration...
[OK]!
```

І дивимося збережені архіви:

```
RT-01#sh archive
The maximum archive configurations allowed is 10.
The next archive file will be named tftp://192.168.10.24/ 13-17-00.303.txt-
1

Archive # Name
1 tftp://192.168.10.24/ 13-16-56.343.txt-0 < - Most Recent
2
3
4
5
6
7
```

```
8
9
10
```

Видно що створено один архів.

У команди ще одна корисна функція – порівняння архівів.

Зробимо (зберігши конфігурацію) ще один архів і перевіримо їх розходження командою:

```
Router# sh archive config differences (name1) (name2)
```

Приклад:

```
RT-01#sh archive config differences tftp://192.168.10.24/ 13-16-56.343.txt-0 tftp://192.168.10.24/ 13-20-30.647.txt-1
Loading 13-16-56.343.txt-0 from 192.168.10.24 (via Port-channel1):!
[OK - 6663 bytes]
Loading 13-20-30.647.txt-1 from 192.168.10.24 (via Port-channel1):!
[OK - 6663 bytes]
!Contextual Config Diffs:
!No changes were found
```

Розходжень немає – архіви однакові.

Так само існує спосіб відновлення попередньої версії архіву командою:

```
RT-01(config)#configure replace tftp://192.168.10.24/ 13-20-30.647.txt-1
```

Другий спосіб зручніше, тому що дозволяє зробити бекап при кожному збереження конфігурації – а значить і можливість відкотитися до останнього (навіть десяти останнім) конфігураціям, але його мінус – не підтримується старими IOS. Для мене дана проблема не актуально – тому я використовую archive.

9 ситуацій, які можна передбачати з вашим Cisco Router-ом.

1. У вас немає резервної копії вашого cisco router.

Картина така: ваш маршрутизатор Cisco відмовляється працювати, а backupa немає. Завжди за замовчуванням робіть backup.

```
Router# copy running-configuration tftp
```

Резервне копіювання конфігурації IOS убудовано в маршрутизатор нових версій IOS, воно автоматично копіює змінені конфігурації роутера.

Існує також багато сторонніх GUI-додатків, які будуть у певний час для вас архівувати. Наприклад, Kiwi CatTools, ManageEngine OpUtils і PacketTrap pt360 Pro.

2. У вас немає backup операційної системи IOS вашого cisco router.

Cisco маршрутизатор абсолютно марний при неправильному налаштуванні. Ви можете копіювати належне IOS назад на маршрутизатор Cisco, що відправлений до вас від Cisco або перенастроїти інший маршрутизатор Cisco (скажемо старий маршрутизатор), що може зайняти місце зламаному маршрутизатору Cisco, ситуації можуть бути різними.

Резервне копіювання на IOS дуже просте. Просто скопіюйте TFTP на свій сервер, з командою, як ця:

```
Router# copy flash tftp
```

3. Немає запасного апаратного маршрутизатора.

Cisco устаткування надзвичайно надійно. Проте, у ході роботи, коли відбуваються помилки й устаткування виходить із ладу, ви повинні бути готові миттєво замінити апарат. Заміна апаратних засобів повинна бути однієї й тієї ж конфігурації (або конфігурації, що забезпечує ті ж мережі для кінцевих користувачів) і IOS також повинні бути однаковими (або пропонують ті самі функції, як необхідної конфігурації).

4. Не ведеться журнал подій на вашім маршрутизаторі.

У першу чергу для виявлення питань потрібно перевіряти журнал подій. Ви також повинні мати в центральному syslog сховище логів Cisco маршрутизатора. Cisco IOS logging легко настроїти, ви можете використовувати безкоштовно Linux syslog server або купити програму для Windows таку як Kiwi Syslog.

5. Давно не обновляли Cisco IOS.

Як і будь-яка інша операційна система, Cisco IOS періодично має помилки. Пам’ятайте, що одержуючи встаткування з новими версіями IOS, треба зберігати сумісність із діючим устаткуванням. Намагайтеся, щоб ваш Cisco IOS залишався актуальним.

6. Не знаю, де шукати документацію й поради по усуненню неполадок.

Використовуйте пошукові системи, наприклад, Google. Але найкраще для пошуку статей і ключових слів використовувати офіційний сайт www.cisco.com

7. Забули пароль і не знаєте як скинути його.

У якийсь момент може трапитися так, що ви забули пароль на ваш маршрутизатор.

8. Не стоїть захист на Cisco маршрутизаторі.

Якщо не подбає про безпеку маршрутизатора й мережі, це може привести в одні з моментів до втрат даних для компанії.

9. Не документуєте дані.

Більшість із нас не любить створювати документації, але про багато речей згодом ми забуваємо й багато хто з нас професійно росте. Документація – зручний спосіб покрокового пояснення більше молодшим адміністраторам, та й служить часом шпаргалкою в роботі для нас самих.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки 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 випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти 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 кібербезпеки 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 кібербезпеки 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

Апаратна архітектура маршрутизаторів Cisco Systems

Маршрутизатор можна розглядати як спеціалізований комп'ютер, що призначений для виконання цілком певних завдань. І як усякий комп'ютер, маршрутизатор має власний центральний процесор (Central Processing Unit – CPU), тип якого може розрізнятися залежно від класу маршрутизатора, фірми-виготовлювача, серії маршрутизатора усередині класу. Основне завдання процесора маршрутизатора (поряд з багатьма другорядними) полягає в обробці вхідних пакетів для прийняття рішень про їхній подальший перенапрямок. При цьому швидкість, з якою маршрутизатор здатний обробляти вступників пакети, прямо залежить від типу використовуваного процесора.

Іншою важливою частиною маршрутизатора, крім процесора, є його пам'ять, що поділена за функціональним принципом. Маршрутизатори компанії Cisco Systems підтримують чотири основних типи пам'яті: постійний запам'ятовувальний пристрій (Read-Only Memory – ROM), флеш-пам'ять (flash memory), пам'ять із довільним доступом (Random-Access Memory – RAM) і енергонезалежну пам'ять (Non-Volatile RAM – NVRAM). З перерахованих типів пам'яті тільки RAM є енергозалежною, тобто її вміст стирається після вимикання живлення маршрутизатора. Тому пам'ять RAM може використовуватися тільки для зберігання тимчасових даних при роботі маршрутизатора.

Пам'ять ROM застосовується для зберігання завантажувального програмного забезпечення (bootstrap software), що запускається першим у момент включення маршрутизатора й надалі відповідає за його завантаження. Деякі типи маршрутизаторів зберігають всю операційну систему Cisco IOS (Internetwork Operating System) у цій пам'яті на випадок виникнення збійних ситуацій, коли

інші джерела, що зберігають образ операційної системи, можуть стати недоступними.

Основне призначення флеш-пам'яті полягає в зберіганні образу (image) операційної системи (власно операційної системи Cisco IOS), що і забезпечує роботу маршрутизатора (у тому випадку, якщо маршрутизатор має таку пам'ять). Адміністратор може зберігати в цій пам'яті образи декількох операційних систем, щоб мати можливість вибрати тип операційної системи при завантаженні маршрутизатора. Розглянутий тип пам'яті реалізується або на процесорній платі маршрутизатора, або на карті PCMCIA.

Крім пам'яті й процесора всі маршрутизатори мають інтерфейси (interfaces, часто в технічній літературі зустрічається термін «порт маршрутизатора»), які обов'язково пойменовані й пронумеровані. При цьому повне ім'я інтерфейсу маршрутизатора містить його тип (наприклад, Serial) і номер, відлічуваний з нуля. На тих маршрутизаторах, які мають попередньо встановлену фіксовану кількість інтерфейсів, нумерація інтерфейсів здійснюється відповідно до їхнього фізичного розташування (порядком проходження) на корпусі маршрутизатора. Так, наприклад, посилання на інтерфейс Ethernet0 мають на увазі посилання на перший інтерфейс локальної мережі. На маршрутизаторах, які дозволяють виконувати зміну інтерфейсів під час його роботи (Online Insertion and Removal – OIR), повне ім'я інтерфейсу містить як мінімум два числа, розділених символом /, де перше число визначає номер слота, у який установлюється інтерфейсний модуль, а друге є номером порту. Наприклад, ім'я інтерфейсу Ethernet5/0 указує на перший інтерфейс Ethernet у п'ятому слоті маршрутизатора. Нагадаємо, що відлік інтерфейсів починається з нуля.

Крім інтерфейсів локальних (Ethernet, Token Ring і т.п.) і глобальних (Serial, ISDN) мереж, всі маршрутизатори компанії Cisco Systems мають консольний порт, що надає асинхронне з'єднання EIA/TIA-232. Такий порт дозволяє за допомогою підключення через консольний кабель управляти

маршрутизатором з комп'ютера. Для зручності підключення на стороні маршрутизатора порт обладнаний роз’ємом JLL45. Подібний спосіб підключення дуже зручний у тому випадку, якщо можливо фізичний доступ до маршрутизатора, а також якщо необхідно провести початкове налаштування нового маршрутизатора або маршрутизатора, у якого всі попередні налаштування були скинуті. У комплекті з новими маршрутизаторами компанія Cisco Systems поставляє консольний кабель. Кабель має достатню довжину, для того щоб при приєднанні до комп'ютера можна було зручно розташувати маршрутизатор, скажемо, на сусідньому столі. Точніше, поставляється навіть не сам кабель, а так званий serial cable console kit – набір, що складається з кабелю й з'єднувачів (connector) різного типу. Такий набір дозволяє підключатися до консольного порту будь-якого маршрутизатора, зробленого компанією Cisco Systems, і іншому встаткуванню (наприклад, до комутаторів). У тому випадку, якщо кабелю в комплекті немає (наприклад, коли був куплений колишній у вживанні маршрутизатор без фірмового впакування), те набір можна просто замовити.

Крім консольного порту, більшість моделей мають порт AUX (auxiliary port – допоміжний порт), що, так само як і консольний, є асинхронним з'єднанням EIA/TIA-232 і використовується для керування маршрутизатором через звичайний модем. Зручність такого керування полягає в тому, що у випадку виникнення проблем з каналами зв'язку на віддалених об'єктах завжди існує можливість одержання доступу до маршрутизатора для виконання тих або інших налаштувань. Як інший приклад можна привести ситуацію, коли адміністратор при керуванні віддаленим маршрутизатором зробив некоректні налаштування, результатом яких виявилося те, що зв'язок на мережному рівні став недоступним. У цьому випадку наявність модему, підключеного до порту AUX, допоможе виправити наслідки допущених помилок. Помітимо, що нові маршрутизатори поставляються з інструкцією (англійською мовою), у якій розписані покрокові дії для підключення модему до AUX-порту.

Важливою складовою частиною маршрутизатора, крім його апаратних компонентів, є конфігураційні файли (configuration files). Конфігураційні файли будуть розглянуті нижче більш детально, а поки обмежимося їхнім коротким перерахуванням. Існує два типи конфігурації операційної системи Cisco IOS: робоча (running configuration) і завантажувальна (startup configuration). Часто конфігурацію першого типу також називають активною (active), так як вона розташовується в оперативній пам'яті маршрутизатора (RAM) і визначає його поточні налаштування. Коли адміністратор виконує команди конфігурування, на маршрутизаторі змінюється вміст саме цієї конфігурації. На противагу робочій (активній), завантажувальна конфігурація розміщається в пам'яті NVRAM маршрутизатора й містить команди операційної системи Cisco IOS, які виконуються в момент його завантаження.

Робоча й завантажувальна конфігурації в якомусь ступені самостійні. Звичайно адміністратор мережі, виконавши початкові етапи налаштування маршрутизатора й перевіривши його працездатність, копіює робочу конфігурацію до пам’яти NVRAM, формуючи в такий спосіб завантажувальну конфігурацію. Очевидно, що основною причиною такої послідовності дій є необхідність збереження зроблених змін при перезавантаженні маршрутизатора.

Початкове налаштування маршрутизатора

Для виконання початкового налаштування маршрутизатора абсолютно необхідно попередньо одержати повну функціональну схему розподіленої мережі. Більше того, якщо мережа охоплює більше двох об'єктів, то без детального промальовування мережі на папері або за допомогою спеціалізованих програм налаштування маршрутизаторів може виявитися дуже обтяжною справою. При цьому важливо не тільки намалювати функціональну схему мережі, але й розробити адресну схему.

Для того щоб опис команд, що приводиться, не був чисто теоретичним, скористаємося прикладами інформаційних повідомлень, отриманих з «живих» маршрутизаторів, які встановлені у вже працюючій мережі однієї з

кіровоградських компаній. Мережа порівняно велика – вона зв'язує більше десяти віддалених об'єктів по виділених каналах зв'язку й по мережі Frame Relay, має вихід в Інтернет і обладнана сервером доступу для обслуговування віддалених користувачів. У мережі як апаратне забезпечення розгорнуте устаткування компанії Cisco Systems – в основному це маршрутизатори для середніх і малих офісів. Можна сміло сказати, що перераховані моделі маршрутизаторів є найпоширенішими, так як поряд з невисокою вартістю вони пропонують найширший спектр можливостей, частина з яких залишається навіть незатребуваною в повсякденній роботі.

Першим кроком налаштування маршрутизатора є вибір його імені, що повинне бути унікальним.

Найбільш зручним є вказівка в імені маршрутизатора даних про його розташування й модель, що нерідко допомагає адміністраторові мережі, так як дозволяє не запам'ятовувати використовувані паролі доступу до маршрутизатора – з його моделі й розташування можна легко визначити пароль, якщо, звичайно, адміністратор вибрав таку схему призначення паролів.

Взагалі обране ім'я може бути абсолютно довільним, однак потрібно додержуватися рекомендацій, наведеним у документі RFC 1035. Ім'я маршрутизатора не повинне займати більше 63 знакомісць. Його варто починати із символу, а закінчуватися воно може або цифрою, або символом. При формуванні імені варто врахувати, що воно є частиною запрошення на уведення команд маршрутизатора, а в запрошенні буде використано не більше 29 символів імені.

У роботі будь-якої розподіленої мережі саму активну участь приймає протокол мережного рівня, причому, як правило, цей протокол є маршрутизуємим. Природно, на маршрутизаторах необхідно проводити його налаштування. Підтримка ж того або іншого протоколу залежить від можливостей операційної системи. Наприклад, якщо використовується операційна система Cisco IP Feature Set, то, як видно з її назви, маршрутизатор

буде підтримувати тільки протокол IP. Якщо ж використовується Cisco Enterprise Feature Set, то буде доступний будь-який маршрутизуємий протокол.

Можна цілком виразно сказати, що підтримки тільки одного протоколу IP цілком достатньо для більшості мереж, адміністратори яких прагнуть до уніфікації програмної бази. Іншою досить вагомою причиною такого вибору є той факт, що з новими маршрутизаторами поставляється саме операційна система IP Feature Set.

У процесі виконання початкового налаштування маршрутизатора варто задати паролі, за допомогою яких надалі буде організований доступ. Символи доступу чутливі до регістра (case sensitive) і можуть містити будь-яку комбінацію прописних або малих літер, цифр і пробілів (останні не повинні бути першими в паролі). Максимальна довжина пароля обмежена 25 символами, хоча на практиці такі довгі паролі рідко знаходять застосування. Для того щоб налаштувати новий маршрутизатор, необхідно підключити до нього за допомогою консольного порту комп'ютер і налаштувати програму емуляції термінала (terminal emulation software). Підключення виконується досить просто. Будь-який комп'ютер має по вкрай мері два послідовних порти: COM1 і COM 2. Звичайно фізичні роз’єми цих портів мають марку або DB9M (DB8 Male), або DM25M (DB25 Male). На невеликих за розміром маршрутизаторах для організації консольного порту використовуються роз’єми RJ45. Крім того, для цих моделей поставляється готовий набір для підключення до комп'ютера. У набір входить кабель, що має на обох кінцях штекери RJ45, і два спеціальних перехідника, що мають із однієї сторони роз’єми RJ45, а з іншого боку – роз’єми DB9F (DB9 Female) або DB25F (DB25 Female). Для підключення маршрутизатора вибирається вільний Сом-порт на комп'ютері, а потім до нього приєднується кабель із необхідним типом

Після фізичного підключення маршрутизатора до комп'ютера для проведення процедури налаштування запускається програма емуляції термінала. Існує безліч готових програмних пакетів емуляції термінала, і часто вони поставляються прямо з операційною системою. Далі необхідно налаштувати

параметри з'єднання, які за замовчуванням описуються популярною схемою 9600-8N1 (9600 baud, 8 data bits, no parity, 1 stop bits).

Тепер, після підключення маршрутизатора до комп'ютера через консольний порт і налаштування програми емуляції термінала можна включати сам маршрутизатор. При цьому на маршрутизаторі починає виконуватися завантажувальне програмне забезпечення (bootstrap software), що запускає тест самодіагностики (Power-On Self-Test – POST), а потім знаходить завантажувальний пристрій (звичайно це флеш-пам'ять), у якому втримується коректний образ операційної системи Cisco IOS. Якщо включення маршрутизатора виконувалося при запущеній програмі емуляції термінала, то повідомлення про початкове завантаження будуть передаватися на консольний порт.

Файли конфігурації маршрутизатора

Важливим завданням адміністратора мережі крім налаштування маршрутизатора є керування файлами конфігурації. Як уже згадувалося, робоча конфігурація (running configuration) містить налаштування, які є активними в процесі роботи маршрутизатора: коли на маршрутизаторі вводяться команди, вони містяться у файл робочої конфігурації. Так як ця конфігурація утримується в оперативній пам'яті маршрутизатора, вона видаляється при вимиканні останнього. На противагу робочій, завантажувальна конфігурація (startup configuration) зберігається в пам'яті NVRAM, і саме вона зчитується маршрутизатором при його завантаженні. Після того як маршрутизатор завантажився, всі команди, що вводяться, міняють робочу конфігурацію, тому, щоб виключити розбіжність конфігурацій через налаштування, зроблених після перезавантаження маршрутизатора, обидві конфігурації варто привести до однакового виду. Завантажувальної конфігурації може й не бути (наприклад, на нових маршрутизаторах). У цьому випадку при першому включенні маршрутизатор перейде в діалоговий режим налаштування (system configuration dialog). Всі дії по керуванню файлами конфігурації виконуються в

привілейованому режимі. При цьому вони не відрізняються від звичайних операцій над файлами в будь-який іншій операційній системі – це перегляд, відновлення, створення резервних копій, заміна й т.п.

В деяких ситуаціях найбільш простим способом з'ясувати поточне налаштування маршрутизатора є перегляд файлів конфігурації. Команда show доступна в користувальницькому режимі, але через те що конфігураційні файли містять багато важливої інформації, наприклад паролі й налаштування протоколу SNMP (Simple Network Management Protocol), перегляд файлів конфігурації, як уже відзначено, можливий тільки в привілейованому режимі. Для перегляду робочої конфігурації варто виконати команду show running-config.

```
Cisco2805#show running-config Building configuration...
Current configuration:
! Last configuration change at 16:58:51 GMT Thu Feb 8 2001
version 12.0 no service pad
service timestamps debug uptime service timestamps log uptime service password-encryption !
hostname Cisco805
I
enable secret 5 $l$mlw$zZERJF6HloTce3LgEfy.J.
```

У першому рядку результату виводиться повідомлення Building configuration..., що говорить про те, що операційна система зайнята збором інформації про свої налаштування й підготовку її для подання в зручному виді. Для полегшення сприйняття секції, на які поділені виведені дані, розділяються символом !. Лістинг містить всю інформацію, необхідну для розуміння майбутнього поводження маршрутизатора. Наприклад, видно, що встановлено ім'я маршрутизатора Cisco2805, що потім «відбивається» як частина запрошення. Далі виводиться пароль доступу в привілейований режим (enable secret password), що встановлюється при початковому налаштуванні маршрутизатора. Для зниження ризику неавторизованого доступу пароль представляється в зашифрованому виді.

Конфігурація є текстовим файлом, збереженим у пам'яті АМ. У процесі завантаження маршрутизатора операційна система переглядає пам'ять NVRAM

на предмет наявності цього файлу, а потім копіює його в оперативну пам'ять. Для того щоб переглянути завантажувальну конфігурацію маршрутизатора, потрібно виконати в привілейованому режимі команду show startup-config.

```
Ciscol005#show startup-config Using 879 out of 7506 bytes
I
version 11.2
service timestamps debug uptime service timestamps log uptime service password-encryption
```

Так як завантажувальна конфігурація зберігається у вигляді текстового файлу, готового для відображення на екрані, то при виконанні даної команди немає затримки, що властива команді show running-config. Перший рядок результату роботи команди show startup-config містить розмір текстового файлу й обсяг пам'яті NVRAM (це значення можна перевірити за допомогою команди show version). Якщо початкове налаштування на новому маршрутизаторі ще не виконувалася, то нам'яти NVRAM буде порожньою. У такому випадку як реакцію на команду show startup-config адміністратор одержить від маршрутизатора наступне повідомлення:

```
%% Non-volatile configuration memory has not been set up or has bad checksum.
```

Зміни, які адміністратор мережі вносить у налаштування маршрутизатора, фіксуються в робочій конфігурації. Якщо після внесення цих змін перезавантажити маршрутизатор, то в його пам'ять буде зчитана завантажувальна конфігурація, що приведе до втрати всіх виконаних адміністратором налаштувань. Для того щоб не допустити втрати налаштувань, варто скопіювати робочу конфігурацію в завантажувальну за допомогою команди copy running-config startup-config. У наведеному нижче прикладі спочатку змінюється ім'я маршрутизатора (у робочій конфігурації), а потім поточна робоча конфігурація записується як завантажувальна. Це дозволяє не допустити втрати змін в імені маршрутизатора.

```
Ciscol005(config)#hostname Ci scol005_Nevsk
Ciscol005_Nevsk#copy running-config startup-config
Building configuration..'.
[OK]
```

Існує тільки один спосіб замінити робочу конфігурацію повністю, що полягає в перезавантаженні маршрутизатора. Однак адміністратор може обновляти робочу конфігурацію маршрутизатора уведенням нових команд. Як уже не раз говорилося, ці команди вводяться як простий текст, причому для їхнього уведення існує кілька джерел, крім використання консольного порту: термінальні порти, пам'ять NVRAM, сервер TFTP, флеш-пам'ять і сервер RPC. Хоча джерелом названий сервер TFTP, реально при передачі команд працює однойменний протокол TFTP (Trivial File Transfer Protocol), що дозволяє передавати файли від сервера до клієнта без якої-небудь авторизації. В основі цього протоколу лежить протокол UDP, що не забезпечує надійності пі в доставці даних, ні в засобах керування потоком трафіку. Щоб скористатися механізмом відновлення конфігурації маршрутизатора, необхідно встановити в мережі сервер, що підтримує цей протокол, а маршрутизатор буде відігравати роль клієнта, що посилає серверу запити на файли.

Використання сервера RPC також дозволяє здійснювати передачу файлів від сервера до клієнта, однак його відмінність від протоколу TFTP полягає в тому, що в цьому випадку як транспортний протокол працює протокол TCP, що забезпечує надійну доставку. Крім того, при передачі потрібно здійснювати автентифікацію, що підвищує безпеку. Використання протоколу TCP дозволяє застосовувати даний метод відновлення конфігурації маршрутизатора через мережі, у яких існує ймовірність виникнення перевантажень і, як наслідок, видалення пакетів. Наприклад, це може спостерігатися в мережі Frame Relay при її істотному завантаженні.

Для уведення команд, що впливають на конфігурацію маршрутизатора, варто перейти в глобальний режим за допомогою команди configure terminal. На додаток до простого уведення команд із клавіатури адміністратор мережі може скористатися можливостями буфера обміну. Якщо існує текстовий файл із командами конфігурації, збережений на локальному диску комп'ютера, з якого здійснюється керування маршрутизатором, то можна просто передати його на

маршрутизатор за допомогою програми емуляції термінала. Команди будуть виконуватися в порядку їхнього розташування у файлі.

Як уже згадувалося, пам'ять NVRAM містить завантажувальний файл конфігурації, що представляє собою текстовий файл із командами, що формують робочу конфігурацію. Адміністратор мережі має право за допомогою команди startup-config running-config обновити робочу конфігурацію маршрутизатора.

На сервері TFTP можуть перебувати текстові файли, команди маршрутизатора або резервна копія робочої конфігурації. Для відновлення робочої конфігурації за допомогою протоколу TFTP передбачена команда tftp J running-config, що пропонує маршрутизатору брати рядки (команди) з текстового файлу на сервері TFTP і поміщати їх у робочу конфігурацію у порядку читання.

```
Router#copy tftp running-config  1
Host or network configuration file [host]? host     |
IP address of remote host [255.255.255.255]? 192.168.3.20    j
Name of configuration file [ Router-config]? commands.txt |
Configure using commands.txt from 192.168.3.20? [confirm] y  J
Loading commands.txt from 192.168.3.20 (via Ethernet): !!!!!     !
[OK - 2035/32723 bytes]
Router#
```

У процесі покрокового виконання даної команди, як видно із приклада, маршрутизатор запитує тип конфігураційного файлу (host or network : configuration file). Конфігураційний файл називається host, якщо він містить команди, які специфічні для певного маршрутизатора. У тому випадку, якщо файл містить у собі команди, загальні для безлічі маршрутизаторів, він називається network. Далі за замовчуванням передбачається, що ім'я файлу складається з ім'я маршрутизатора й наступного за ним слова – config. Потім маршрутизатор запитує адресу IP-сервера, на якому працює програмне забезпечення, що підтримує функції сервера TFTP. Замість вказівки адреси адміністратор може ввести ім'я пристрою. Тоді маршрутизатор скористається механізмом дозволу імен для одержання адреси з ім'я. Після підтвердження відновлення маршрутизатор завантажує зазначений файл із сервера, відзначаючи

успішно скопійовані блоки символами !. Для того щоб уникнути помилок наприкінці копіювання, необхідно в кінець текстового файлу з командами помістити команду end, що необхідна для виходу з конфігураційного режиму при закінченні копіювання. Команда copy tftp running-config була документована у версії 11.0 операційної системи, а оригінальна команда відновлення робочої конфігурації із сервера TFTP має ім'я configure network.

Для того щоб обновити робочу конфігурацію за допомогою сервера RPC, адміністратор мережі може скористатися командою copy RPC running-config, що вказує маршрутизатору зчитувати рядки текстового файлу на сервері й поміщати їх у робочу конфігурацію, так, ніби вони вводилися вручну. Аналогічно случаю із сервером TFTP, останньою командою й текстовим файлом повинна бути команда end. Діалог, що веде маршрутизатор з адміністратором мережі, подібний до діалогу у випадку із сервером TFTP. Але оскільки протокол RPC вимагає проведення автентифікації для виконання копіювання, буде потрібно ще кілька кроків, щоб маршрутизатор одержав доступ до файлів на сервері. Так, на маршрутизаторі необхідно задати ім'я користувача, що буде вказуватися при доступі, за допомогою команди ip rcmd remote-username. Далі на сервері RPC потрібно створити користувача з аналогічним ім'ям і задати йому домашній каталог на сервері, у якому повинні розміщатися текстові файли конфігурації. Після цього на сервері в домашньому каталозі користувача варто створити файл (. rhost), що містить рядка для кожного пристрою й облікового запису, яким дозволяється доступ до цього каталогу. Але, загалом кажучи, рекомендується звернутися до довідкової системи програмного забезпечення сервера RPC, так як розгляд деталей його налаштування виходить за рамки книги.

Текстові файли, які були збережені у флеш-пам'яті, також підходять для відновлення робочої конфігурації. При цьому джерелом може бути внутрішня флеш-пам'ять або пам'ять, організована на карті PCMCIA, вставлену в

однойменний слот маршрутизатора. При копіюванні текстового файлу варто вказати в команді копіювання пристрій і ім'я файлу.

Адміністратор мережі повинен розглянути питання про створення резервних копій конфігураційних файлів маршрутизатора. При цьому в його розпорядженні є декілька одержувачів, яким можна направити резервну копію. Одержувачі відповідають джерелам, які були розглянуті вище, за винятком локального диска. При створенні резервної копії маршрутизатор будує конфігурацію й форматує її як звичайний текстовий файл, що потім копіюється зазначеному одержувачеві. При цьому всі операції аналогічні операціям, пов'язаним з відновленням конфігурації маршрутизатора.

Для того щоб створити резервну копію робочої конфігурації в пам'яті NVRAM, варто скористатися командою copy running-config startup-config, виконання якої приводить до заміни завантажувальної конфігурації маршрутизатора його робочою конфігурацією. Дана команда повинна виконуватися щораз після зміни налаштувань маршрутизатора, щоб при наступному його завантаженні зроблені зміни залишилися в силі. Ця команда була вперше документована у версії 11.0 операційної системи Cisco IOS. Оригінальна команда для копіювання робочої конфігурації до пам’яти NVRAM – write memory.

Для того щоб скопіювати робочу конфігурацію в зазначений файл на сервер TFTP, у розпорядженні адміністратора є команда copy running-config tftp. Нижче наведена приклад роботи команди. Ця команда була вперше документована у версії 11.0 операційної системи Cisco IOS. Оригінальна команда для створення резервної копії робочої конфігурації на сервері TFTP – команда write network.

```
Router#copy running-config tftp
Remote host [192.168.3.20]?
Name of configuration file to write [ router-confg]?
Write file router-confg on host 192.168.3.20? [confirm] в
#
```

```
Writing router-confg!!! [OK]
```

Створення резервної копії на сервері RPC виконується за аналогією з попереднім прикладом, за винятком того, що варто користуватися командою running-config RPC, що створює текстовий файл на зазначеному сервері RPC. Щоб створити копію робочої конфігурації у флеш-пам'яті, для маршрутизаторів серії Cisco 1600 адміністратор може вдатися до команди copy running-config device: < номер розділу: > <ім'я файлу>. Слід зазначити, що маршрутизатор не в змозі записати інформацію у флеш-пам'ять, якщо остання має статус «тільки для читання» (read-only). Перегляд статусу пам'яті виробляється командами show version або show flash.

```
Ciscol005#show flash
PCMCIA flash directory:
File Length Name/status 1   1589513  Cl 005-y-mz.11 3-7.bin
[1589580 bytes used. 507572 available. 2097152 total]
2048K bytes of processor board PCMCIA flash (Read/Write)
Ciscol601#show flash
PCMCIA flash directory:
File Length Name/status 1   4061104 cl 600-y-l.113-3.T.bin
[4061168 bytes used. 2230288 available. 6291456 total]
6144K bytes of processor board PCMCIA flash (Read ONLY)
```

Команди для заміни завантажувальної конфігурації аналогічні тим, які використовуються для робочої конфігурації. При цьому необхідно замінити ключове слово running-config на startup-config. Однак команди, які працюють із завантажувальною конфігурацією, мають одну відмінність, що складається в тому, що вони заміняють зазначеним файлом уміст пам'яті NVRAM, у той час як команди, що впливають на робочу конфігурацію, заміняють команди в активній конфігурації маршрутизатора.

Найбільше часто адміністраторові мережі доводиться заміняти завантажувальну конфігурацію робочої при проведенні в останньій будь-яких змін. Для цього є команда, що вже розглядалася докладно раніше, а саме running-confug startup-config. Нагадаємо, що ця команда заміняє вміст пам'яті NVRAM файлом робочої конфігурації за умови, що в цій пам'яті досить вільного місця.

Нагадаємо також, що оригінальна команда операційної системи Cisco IOS для заміни завантажувальної конфігурації робочою – write memory.

Щоб замістити вміст пам'яті NVRAM файлом із сервера TFTP, можна скористатися командою tftp startup-config, виконання якого аналогічно роботі із сервером TFTP. Адміністраторові мережі перед її виконанням варто довідатися адресу серверу й ім'я файлу для копіювання. Операційна система маршрутизатора дозволить скопіювати до пам’яти NVRAM будь-який файл, Навіть якщо він не містить конфігураційних команд, хоча в останньому випадку відбудеться збій при завантаженні маршрутизатора. Розглянута команда була вперше документована у версії 11.0 операційної системи Cisco IOS, а оригінальна команда створення копіювання файлу до пам’яти NVRAM – configure 1 overwrite-network.

Іноді буває необхідно почати конфігурацію «з нуля». Коли маршрутизатор стартує без завантажувальної конфігурації, його операційна система припускає, що він не був налаштований і переходить у програму покрокового налаштування (System Configuration Dialog). Тобто, видаливши завантажувальну конфігурацію маршрутизатора, можна почати заново налаштовувати маршрутизатор. Щоб видалити вміст пам'яті NVRAM, адміністратор мережі може виконати команду erase startup-config. Мабуть, найбільше застосування ця команда знаходить у лабораторних (тестових) умовах, коли досить часто потрібно повністю міняти конфігурацію маршрутизатора. Команда була вперше задокументова на у версії 11.0 операційної системи Cisco IOS. Оригінальна команда для видалення вмісту пам'яті NVRAM – write erase. Слід зазначити, що якщо маршрутизатор настроюється за допомогою програми Cisco ConfigMaker, то при відновленні налаштувань маршрутизатора його стара конфігурація буде видалятися автоматично.

3.2 Розробка структурної схеми

Структурна схема роботи системи зображена на рисунку 3.1. З неї можна побачити навіщо необхідно розроблене бакалаврське ПЗ. На налаштованій локальній мережі відбувається регулярне резервного копіювання налаштувань маршрутизаторів до бази даних за протоколом TFTP.

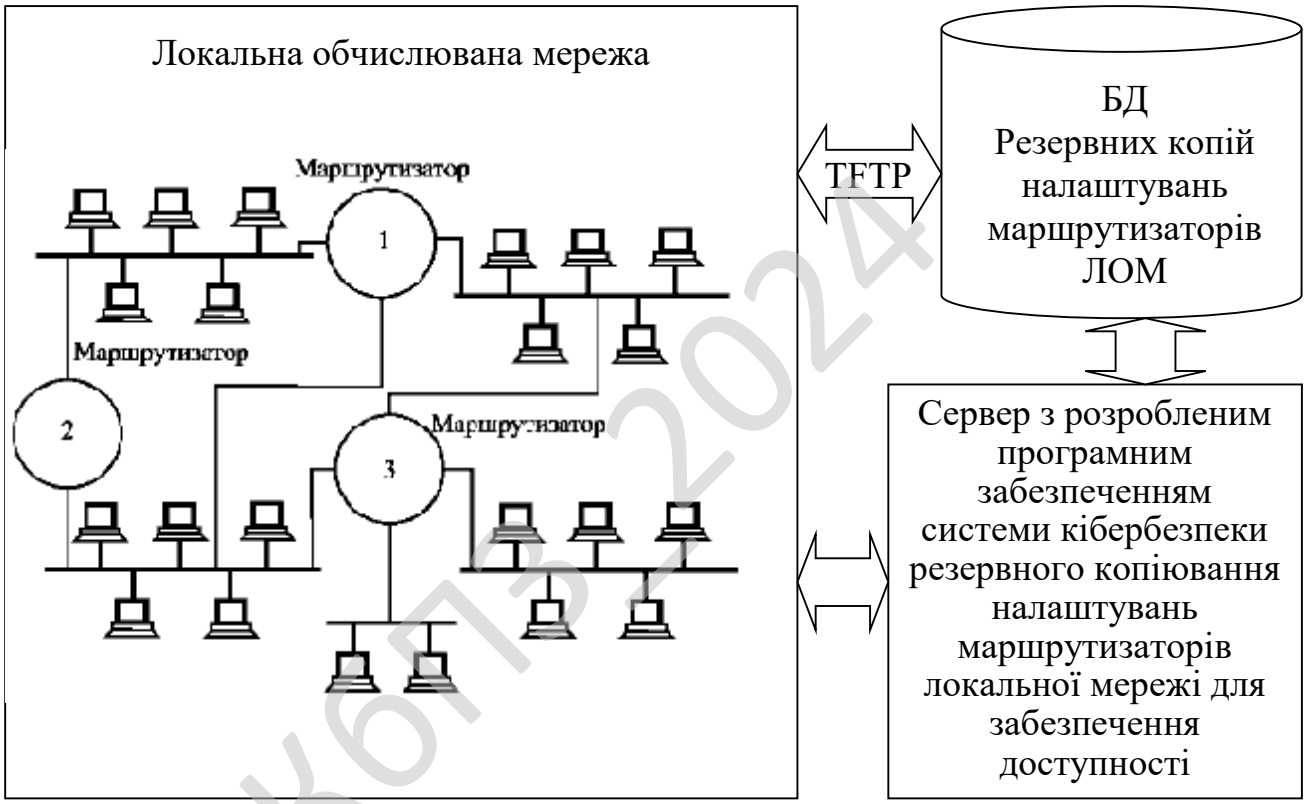

Рисунок 3.1 – Структурна схема системи

Адміністратор встановлює розроблене бакалаврське ПЗ на довільний сервер, як наведено на схемі це файловий сервер. За допомогою розробленого ПЗ адміністратор може перевіряти всі маршрутизатори з заданим інтервалом часу та при виведенні з ладу маршрутизатора (атаковане устаткування не може відповісти її користувачам) провести перезавантаження обладнання.

Розроблене ПЗ дозволяє захищати маршрутизатори локальних мереж від деструктивних дій програмних та апаратних помилок.

Захист проводиться інтервальними перевірками налаштувань маршрутизаторів ЛОМ, при виявленні помилок проходить автоматичне переналаштування маршрутизатора до нормальних значень через резервне копіювання даних по протоколу Trivial File Transfer Protocol (TFTP).

Опис протоколу TFTP

TFTP використовується головним чином для первісної завантаження бездискових робочих станцій. TFTP, на відміну від FTP, не містить можливостей автентифікації (хоча можлива фільтрація по IP-адресою) і заснований на транспортному протоколі UDP.

Основне призначення TFTP – забезпечення простоти реалізації клієнта. У зв'язку з цим він використовується для завантаження бездискових робочих станцій, завантаження оновлень і конфігурацій в "розумні" мережеві пристрої, записи статистики з міні-АТС (CDR) і апаратних маршрутизаторів / файрволов.

Оскільки протокол не підтримує авутентифікації, єдиний метод ідентифікації клієнта – це його мережеву адресу (який може бути підроблений). Зазвичай в Unix-системах tftpd доступний тільки каталог / tftpboot. Однак у старих TFTP-серверах було можливим отримати файл паролів командою RRQ .. / etc / passwd.

Демон tftpd (одна з реалізацій tftp-сервера) відмовляється обробляти файли, що містять у своєму імені комбінацію "/ .. /" або що починається з ".. /". Запис дозволяється тільки в файли, які вже існують (будь-якого розміру, наприклад нульового), і доступні для публічної запису (права доступу: -rw).

Додатковий захист від доступу до довільних файлів здійснюється за допомогою зміни кореневого каталогу на каталог tftpd (зазвичай / usr / TFTPRoot).

Типи пакету

Спочатку в TFTP-пакеті йде поле розміром в 2 байта, визначальне тип пакету:

Read Request (RRQ, # 1) – запит на читання файлу.

Write Request (WRQ, # 2) – запит на запис файлу.

Data (DATA, # 3) – дані, передані через TFTP.

Acknowledgment (ACK, # 4) – підтвердження пакета.

Error (ERR, # 5) – помилка.

Запити на читання і запис

Для початку передачі даних клієнт повинен послати сервера WRQ або RRQ-пакет. В обох пакетів формат однаковий.

Таблиця 3.1 – Запити на читання і запис

0x01/0x02 (тип пакету)	Ім'я файлу	0x00 (кінець рядка)	Режим передачі	0x00 (кінець рядка)	Опції ... (якщо є)
2 байта	рядок в ASCII	1 байт	рядок в ASCII	1 байт	Див "Опції"

У TFTP існує 2 режими передачі (режим Mail, визначений в IEN 133, визнаний застарілим):

– netascii – файл перед передачею перекодовується в ASCII.

– octet – файл передається без змін.

Після отримання RRQ-пакета сервером, він відразу починає передачу даних. У випадку з WRQ-запитом – сервер повинен надіслати ACK-пакет с номером пакету 0.

Процес передачі даних

Після отримання запиту RRQ, сервер відразу посилає в якості підтвердження пакет з даними і з ID пакета рівним одиниці. У WRQ в якості підтвердження використовується ACK з ID рівним нулю. Всього по TFTP можна

передати 32 Мб (65536 * 512/1024), однак через використання знакового int замість беззнакового, розмір підтвердження обмежений 16 мегабайтами. Однак якщо клієнт і сервер підтримують розширення протоколу RFC 2347 і RFC 2348, то максимальний розмір переданого файлу збільшується до 4Gb.

Опції TFTP

У RFC 2347 був передбачений формат опцій, які можна приєднувати до закінчення RRQ-пакета і WRQ-пакета.

Таблиця 3.2 – Опції TFTP

Код опції	0x00 (кінець рядка)	Значення опції	0x00 (кінець рядка)
рядок в ASCII	1 байт	рядок в ASCII	1 байт

Опцій може бути кілька. Тоді вони будуть слідувати один за одним. Порядок опцій не важливий.

У відповідь на RRQ (або WRQ) з опціями, сервер повинен надіслати OACK з списком опцій, які сервер прийняв. Найбільш поширені опції.

Таблиця 3.3 – Відповідь на RRQ

Назва	Визначена в	Код опції	
Розмір блоку	RFC 2348	blksize	Як значення опції йде число, що набуває значення від 8 до 65 464, що позначає розмір блоку.
Інтервал повторної передачі (Timeout)	RFC 2349	timeout	Як значення опції йде число, що набуває значення від 1 до 255, що позначає час очікування перед повторною передачею блоку в секундах.
Розмір файлу	RFC 2349	tsize	Як значення опції йде число, що позначає розмір переданого файлу в байтах.

Помилки

У TFTP інформація про помилку має наступний формат:

Таблиця 3.4 – Інформація про помилку

0x05 (тип пакету)	Код помилки	Опис помилки	0x00 (кінець рядка)
2 байта	2 байти	рядок в ASCII	1 байт

Код помилки може приймати одне із значень, перерахованих в STD 33 (за винятком коду 8 – він описаний в RFC 2347). Ось вони:

Таблиця 3.5 – Код помилки

Код помилки	Опис
0	Немає певного коду, див. текст помилки
1	Файл не знайдено
2	Доступ заборонено
3	Неможливо виділити місце на диску
4	Некоректна TFTP-операція
5	Неправильний Transfer ID
6	Файл вже існує
7	Користувач не існує
8	Неправильна опція

Схема URI

У RFC 3617 визначено формат URI для TFTP. Він має наступний вигляд:

tftp :/ / [вузол призначення] / [потрібний файл]; mode = [режим передачі]

Наприклад:

tftp :/ / example.com / todo.txt; mode = netascii

Розроблене ПЗ буде вкрай актуально адміністраторам при наявності в мережі великої кількості маршрутизаторів.

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема роботи системи. Для реалізації програми наряду з протоколом TFTP використовується протокол RPC.

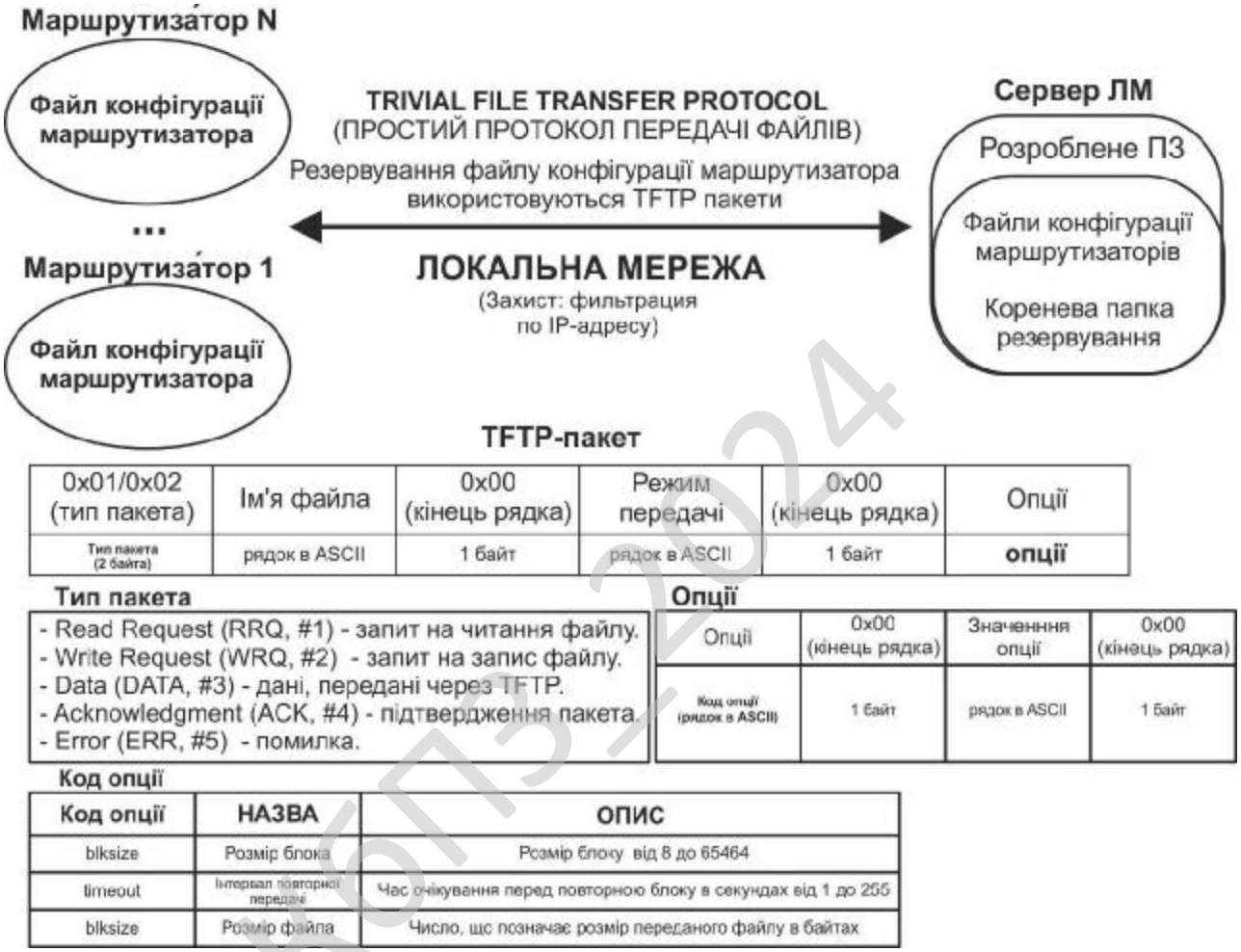

Рисунок 3.2 – Функціональна схема системи

Виклик віддалених процедур (англ. Remote procedure call, RPC) – протокол, що дозволяє програмі, запущеній на одному комп'ютері бути викликаною на іншому комп'ютері без написання безпосередньо коду для цієї операції.

Концепція віддаленого виклику процедур

Ідея виклику віддалених процедур (англ. Remote Procedure Call – RPC) полягає у розширенні механізму передачі управління і даних усередині програми, що виконується на одній машині, на передачу керування й даних через мережу. Засоби віддаленого виклику процедур призначені для полегшення організації розподілених обчислень. Найбільша ефективність використання RPC досягається в тих додатках, в яких існує інтерактивний зв'язок між віддаленими компонентами з невеликим часом відповідей і відносно малою кількістю переданих даних. Такі програми називаються RPC-орієнтованими.

Існує ряд технологій, що забезпечують RPC, зокрема:

– Sun RPC (бінарний протокол на базі TCP та UDP).

– Net Remoting (бінарний протокол на базі TCP, UDP, HTTP).

– XML-RPC (текстовий протокол на базі HTTP).

– SOAP – Simple Object Access Protocol (текстовий протокол на базі HTTP).

– Java RMI – Java Remote Method Invocation.

– JSON-RPC JavaScript Object Remote Procedure Calls (текстовий, на базі HTTP).

Характерними рисами виклику локальних процедур є:

– Асиметричність, тобто одна із сторін є ініціатором;

– Синхронність, тобто виконання процедури, що викликає віддалену процедуру, призупиняється з моменту видачі запиту і відновлюється тільки після повернення з викликаної процедури.

Реалізація віддалених викликів істотно складніше реалізації викликів локальних процедур. Почнемо з того, що оскільки викликаюча і викликана процедури виконуються на різних машинах, то вони мають різні адресні простори, і це створює проблеми при передачі параметрів і результатів, особливо якщо машини не ідентичні. Так як RPC не може розраховувати на розподілену пам'ять, то це означає, що параметри RPC не повинні містити вказівників на дані

в нестековій пам'яті і що значення параметрів повинні копіюватися з одного комп'ютера на інший.

Але в реалізації RPC беруть участь щонайменше два процеси – по одному в кожній машині. У випадку, якщо один з них аварійно завершиться, можуть виникнути такі ситуації: при аварії викликаючої процедури віддалено викликані процедури стануть «осиротілими», а при аварійному завершенні віддалених процедур стануть «знедоленими батьками» викликаючі процедури, які будуть безрезультатно чекати відповіді від віддалених процедур.

Крім того, існує ряд проблем, пов'язаних з неоднорідністю мов програмування та операційних середовищ: структури даних і структури виклику процедур, підтримувані в будь-якому однією мовою програмування, не підтримуються точно так само у всіх інших мовах.

Ці та деякі інші проблеми вирішує широко поширена технологія RPC, що лежить в основі багатьох розподілених операційних систем, наприклад Plan 9.

Базові операції RPC

Щоб зрозуміти роботу RPC, розглянемо спочатку виконання виклику локальної процедури у звичайній машині, що працює автономно. Нехай це, наприклад, буде системний виклик: count = read (fd, buf, nbytes); де fd – ціле число, buf – масив символів, nbytes – ціле число.

Щоб здійснити виклик, викликаюча процедура заштовхує параметри в стек у зворотному порядку. Після того, як виклик read виконаний, він поміщає значення, що повертається в регістр, переміщує адресу повернення і повертає управління викликаючій процедурі, яка вибирає параметри з стека, повертаючи його в початковий стан. Зауважимо, що в мові С параметри можуть викликатися або по посиланню (by name), або за значенням (by value). По відношенню до викликаючої процедури параметри-значення є ініціалізованими локальними змінними. Викликана процедура може змінити їх, і це не вплине на значення оригіналів цих змінних в викликаючій процедурі. Якщо в визвану процедуру передається покажчик на змінну, то зміна значення цієї змінної визваної

процедури тягне зміну значення цієї змінної і для викликаючої процедури. Цей факт дуже істотний для RPC.

Існує також інший механізм передачі параметрів, який не використовується в мові С. Він називається call-by-copy/restore і полягає в необхідності копіювання викликаючою програмою змінних в стек у вигляді значень, а потім копіювання назад після виконання виклику поверх оригінальних значень викликаючої процедури.

Рішення про те, який механізм передачі параметрів використовувати, приймається розробниками мови. Іноді це залежить від типу переданих даних. У мові С, наприклад, цілі та інші скалярні дані завжди передаються за значенням, а масиви – по посиланню.

Ідея, покладена в основу RPC, полягає в тому, щоб зробити виклик віддаленої процедури по можливості схожим до виклику локальної процедури. Іншими словами – зробити RPC прозорим: викликаючій процедурі не потрібно знати, що викликається процедура знаходиться на іншій машині, і навпаки.

RPC досягає прозорості наступним шляхом. Коли викликана процедура дійсно є віддалена, в бібліотеку поміщається замість локальної процедури інша версія процедури, яка називається клієнтським стабом (stub – заглушка). Подібно оригінальної процедурі, стаб викликається з використанням викликаючої послідовності, так само відбувається переривання при зверненні до ядра. Тільки на відміну від оригінальної процедури він не поміщає параметри в регістри і не запитує у ядра дані, замість цього він формує повідомлення для відправки ядру віддаленої машини.

Етапи виконання RPC

Після того, як клієнтський стаб був викликаний програмою-клієнтом, його першим завданням є заповнення буфера відправлені повідомленням. У деяких системах клієнтський стаб має єдиний буфер фіксованої довжини, що заповнюється щоразу з самого початку при вступі кожного нового запиту. В інших системах буфер повідомлення являє собою пул буферів для окремих полів

повідомлення, причому деякі з цих буферів вже заповнені. Цей метод особливо підходить для тих випадків, коли пакет має формат, що складається з великої кількості полів, але значення багатьох з цих полів не змінюються від виклику до виклику.

Потім параметри повинні бути перетворені у відповідний формат і вставлені в буфер повідомлення. До цього моменту повідомлення готове до передачі, тому виконується переривання за викликом ядра.

Коли ядро отримує управління, воно перемикає контексти, зберігає регістри процесора і карту пам'яті (дескриптори сторінок), встановлює нову карту пам'яті, яка буде використовуватися для роботи в режимі ядра. Оскільки контексти ядра і користувача розрізняються, ядро має точно скопіювати повідомлення в свій власний адресний простір, так, щоб мати до нього доступ, запам'ятати адресу призначення (а, можливо, і інші поля заголовка), а також воно має передати його мережевому інтерфейсу. На цьому завершується робота на клієнтській стороні. Включається таймер передачі, і ядро може або виконувати циклічне опитування наявності відповіді, або передати управління планувальником, який обере будь-який інший процес на виконання. У першому випадку прискорюється виконання запиту, але відсутнє мультипрограмування.

На стороні сервера біти, що надходять, поміщаються приймаючої апаратурою або у вбудований буфер, або в оперативну пам'ять. Коли вся інформація буде отримана, генерується переривання. Обробник переривання перевіряє правильність даних пакета і визначає, якому стабу слід їх передати. Якщо жоден із стабів не очікує цей пакет, обробник повинен або помістити його в буфер, або взагалі відмовитися від нього. Якщо є очікуючий стаб, то повідомлення копіюється йому. Нарешті, виконується переключення контекстів, в результаті чого відновлюються регістри і карта пам'яті, приймаючи ті значення, які вони мали в момент, коли стаб зробив виклик receive.

Тепер починає роботу серверний стаб. Він розпаковує параметри і поміщає їх відповідним чином в стек. Коли все готово, виконується виклик

сервера. Після виконання процедури сервер передає результати клієнту. Для цього виконуються всі описані вище етапи, тільки в зворотному порядку.

14 етапів виконання RPC:

Клієнт:

1. Виклик стабу.
2. Підготувати буфер.
3. Упакувати параметри.
4. Заповнити поле заголовка.
5. Обчислити контрольну суму в повідомленні.
6. Переривання до ядра.
7. Черга пакету на виконання.
8. Передача повідомлення контролеру по шині QBUS.
9. Час передачі по мережі Ethernet.

Сервер:

10. Отримати пакет від контролера.
11. Процедура обробки переривання.
12. Обчислення контрольної суми.
13. Переключення контексту в простір користувача.
14. Виконання серверного стабу.

Динамічне зв'язування

Розглянемо питання про те, як клієнт задає місце розташування сервера. Одним з методів вирішення цієї проблеми є безпосереднє використання адресу сервера в клієнтській програмі. Недолік такого підходу – його надзвичайна негнучкість: при переміщенні сервера, або при збільшенні числа серверів, або при зміні інтерфейсу у всіх цих та багатьох інших випадках необхідно перекомпілювати всі програми, які використовували жорстке завдання адреси сервера. Для того, щоб уникнути всіх цих проблем, в деяких розподілених системах використовується так зване динамічне зв'язування.

Початковим моментом для динамічного зв'язування є формальне визначення (специфікація) сервера. Специфікація містить ім'я файлу-сервера, номер версії і список процедур-послуг, що надаються даним сервером для клієнтів. Для кожної процедури дається опис її параметрів із зазначенням того, чи є даний параметр вхідним чи вихідним щодо сервера. Деякі параметри можуть бути одночасно вхідними і вихідними – наприклад, деякий масив, який надсилається клієнтом на сервер, модифікується там, а потім повертається назад клієнтові.

Формальна специфікація сервера використовується в якості вихідних даних для програми-генератора стабів, яка створює як клієнтські, так і серверні стаби. Потім вони поміщаються у відповідні бібліотеки. Коли користувальницька (клієнтська) програма викликає якусь процедуру, визначену в специфікації сервера, відповідна стаб-процедура зв'язується з двійковим кодом програми.

При запуску сервера найпершою його дією є передача свого серверного інтерфейсу спеціальною програмою, так званим binder'ом. Цей процес, відомий як процес реєстрації сервера, включає передачу сервером свого імені, номера версії, унікального ідентифікатора і описувача місцезнаходження сервера. Описувач системно незалежний і може представляти собою IP, Ethernet, X.500 або ще яку-небудь адресу. Крім того, він може містити й іншу інформацію, наприклад дані, що стосуються автентифікації.

Коли клієнт викликає одну з віддалених процедур перший раз, наприклад, read, клієнтський стаб бачить, що він ще не підключений до сервера, і надсилає повідомлення binder-програмі з проханням про імпорт інтерфейсу потрібної версії потрібного сервера. Якщо такий сервер існує, то binder передає описувач і унікальний ідентифікатор клієнтському стабу.

Клієнтський стаб при посилці повідомлення із запитом використовує в якості адреси описувач. У повідомленні містяться параметри і унікальний ідентифікатор, який ядро сервера використовує для того, щоб направити

надійшло повідомлення в потрібний сервер у випадку, якщо їх декілька на цій машині.

Цей метод, що полягає в імпорті / експорті інтерфейсів, володіє високою гнучкістю. Наприклад, може бути кілька серверів, що підтримують один і той же інтерфейс, і клієнти розподіляються по серверах випадковим чином. У рамках цього методу стає можливим періодичне опитування серверів, аналіз їх працездатності та, у разі відмови, автоматичне відключення, що підвищує загальну відмовостійкість системи. Цей метод може також підтримувати аутентифікацію клієнта. Наприклад, сервер може визначити, що він може бути використаний тільки клієнтами з певного списку.

Однак у динамічного зв'язування є недоліки, наприклад, додаткові накладні витрати (тимчасові витрати) на експорт та імпорт інтерфейсів. Величина цих витрат може бути значною, тому що багато клієнтські процеси існують короткий час, а при кожному старті процесу процедура імпорту інтерфейсу повинна бути знову виконана. Крім того, у великих розподілених системах може бути вузьким місцем програма binder, а створення декількох програм аналогічного призначення також збільшує накладні витрати на створення і синхронізацію процесів.

Семантика RPC в разі відмов

В ідеалі RPC повинен функціонувати правильно і у випадку відмов. Розглянемо наступні класи відмов:

– Клієнт не може визначити місцезнаходження сервера, наприклад, у разі відмови потрібного сервера, або через те, що програма клієнта була скомпільована давно і використовувала стару версію інтерфейсу сервера. У цьому випадку у відповідь на запит клієнта надходить повідомлення, що містить код помилки.

– Втрачено запит від клієнта до сервера. Найпростіше рішення – через певний час повторити запит.

– Втрачено повідомлення-відповідь від сервера до клієнта. Цей варіант складніший від попереднього, так як деякі процедури не є ідемпотентними. Ідемпотентною називається процедура, запит на виконання якої можна повторити кілька разів, і результат при цьому не зміниться. Прикладом такої процедури може бути читання файлу. Але ось процедура зняття деякої суми з банківського рахунку не є ідемпотентною, і в разі втрати відповіді повторний запит може істотно змінити стан рахунку клієнта. Одним з можливих рішень є приведення всіх процедур до ідемпотентного виду. Однак на практиці це не завжди вдається, тому може бути використаний інший метод – послідовна нумерація всіх запитів клієнтським ядром. Ядро сервера запам'ятовує номер самого останнього запиту від кожного з клієнтів, і при отриманні кожного запиту виконує аналіз – чи є цей запит первинним або повторним.

– Сервер зазнав аварію після отримання запиту. Тут також важливо властивість Ідемпотентний, але на жаль не може бути застосований підхід з нумерацією запитів. У цьому випадку має значення, коли відбулася відмова – до чи після виконання операції. Але клієнтське ядро не може розпізнати ці ситуації, для нього відомо тільки те, що час відповіді вичерпано.

Існує три підходи до цієї проблеми:

– Чекати до тих пір, поки сервер не перезавантажиться і намагатися виконати операцію знову. Цей підхід гарантує, що RPC був виконаний до кінця принаймні один раз, а можливо і більше.

– Відразу повідомити додатка про помилку. Цей підхід гарантує, що RPC був виконаний не більше одного разу.

– Третій підхід не гарантує нічого. Коли сервер відмовляє, клієнтові не надається ніякої підтримки. RPC може бути або не виконано взагалі, чи виконано багато разів. В усякому разі цей спосіб дуже легко реалізувати.

На рисунку 3.2 показано як функціонально працює розроблена система з довільним маршрутизатором локальної мережі.

Розроблене ПЗ відносно встановлених налаштувань адміністратора ЛМ проводить опит маршрутизаторів через деякий інтервал часу, при виявлені помилок в роботі проводить перезавантаження маршрутизатора. Це відбувається трьома етапами.

Перший етап це програмний скид маршрутизатора з відключенням від обслуговування всіх портів.

Другий етап завантаження з серверу через протокол Trivial File Transfer Protocol (простий протокол передачі файлів) пакету налаштувань маршрутизатора. Для захисту використовується фільтрація по IP адресі, тобто тільки файловий сервер мережі з відповідною адресою це може проводити.

Третій це подача команди початок обслуговування всіх портів. Якщо маршрутизатор працює у нормальному режимі (без збою) адміністратор ЛМ може подати запит на резервне копіювання поточних налаштувань маршрутизатора.

Для початку передачі даних клієнт повинен послати серверу WRQ або RRQ-пакет (в обох пакетів формат однаковий). У TFTP існує 2 режиму передачі (режим Mail, визначений у IEN 133, визнаний застарілим): netascii – файл перед передачею перекодовується в ASCII. октет – файл передається без змін. Після отримання RRQ-пакета сервером, він відразу починає передачу даних. У випадку з WRQ-запитом – сервер має надіслати ACK-пакет із номером пакета 0.

Розроблене ПЗ через Trivial File Transfer Protocol підтримує п’ять типів пакетів:

– Read Request (RRQ, #1) – запит на читання файлу налаштувань обраного маршрутизатора.

– Write Request (WRQ, #2) – запит на запис файлу налаштувань обраного маршрутизатора.

– Data (DATA, #3) – дані, передані через TFTP.

– Acknowledgment (ACK, #4) – підтвердження пакета.

– Error (ERR, #5) – помилка доступу, тобто маршрутизатор заблоковано.

Кожен пакет має поле опцій, які можна приєднувати до закінчення RRQ-пакета і WRQ-пакету, поля пакету можна переглянути на рисунку 3.2.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.3.

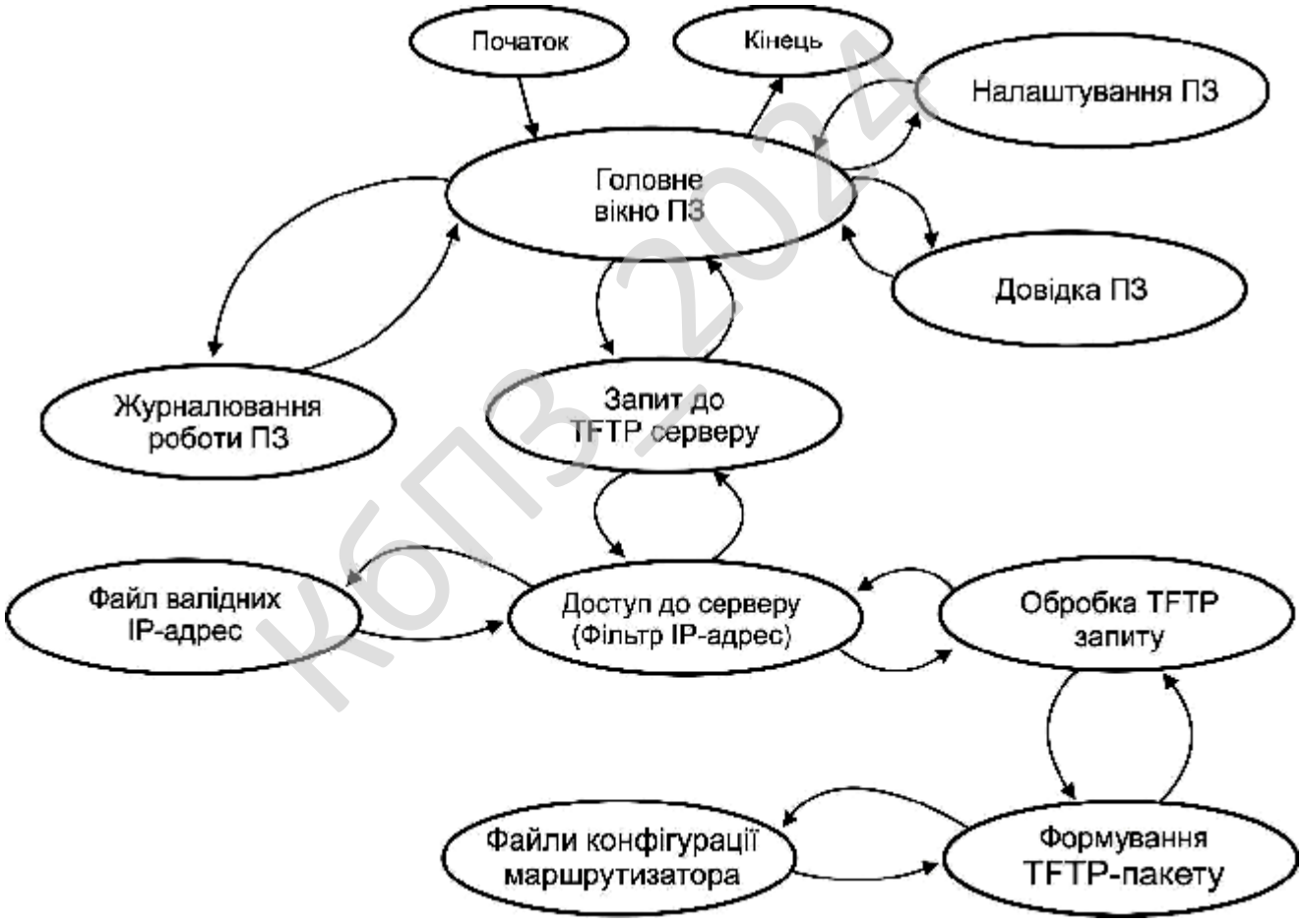

Рисунок 3.3 – Діаграма взаємодії процесів

Після початку роботи ПЗ ми потрапляємо до головного вікна ПЗ звідки можемо переглянути довідку ПЗ, налаштування ПЗ та журналювання роботи ПЗ.

Далі через запит до TFTP серверу потрапити до доступу до серверу який працює на основі фільтру IP-адрес із застосуванням файлу валідних IP-адрес.

Після чого через обробку TFTP запиту проводиться формування TFTP-пакету та отримання файлу конфігурації маршрутизатора.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків:

– Початкова ініціалізація ПЗ.

– Завантаження модулів ПЗ.

– Завантаження збережених налаштувань ПЗ.

– ПЗ завантажено (запит).

– Є доступ до Інтернет (запит).

– Моніторинг TFTP запиту.

– Повідомлення WM_CLOSE (запит).

– Поступив TFTP пакет (запит).

– Пакет TFTPCClose (запит).

– Читання та оновлення файлів налаштувань маршрутизатора.

– Встановлення інтервалів часу перевірки TFTP запитів.

– Пакет TFTPverification (запит).

– Підпрограма оновлення файлу налаштувань маршрутиза́тора.

– Пакет TFTPreserv (запит).

– Отримання поточної дати та часу.

– Формування ім'я файлу копії налаштувань маршрутизатора.

– Запис файлу копії налаштувань маршрутизатора у каталог доступу.

– Запис до журналу роботи ПЗ.

– Повідомлення WM_CLOSE (запит).

– Звільнення виділених динамічних ресурсів ПЗ.

– Передача управління ОС.

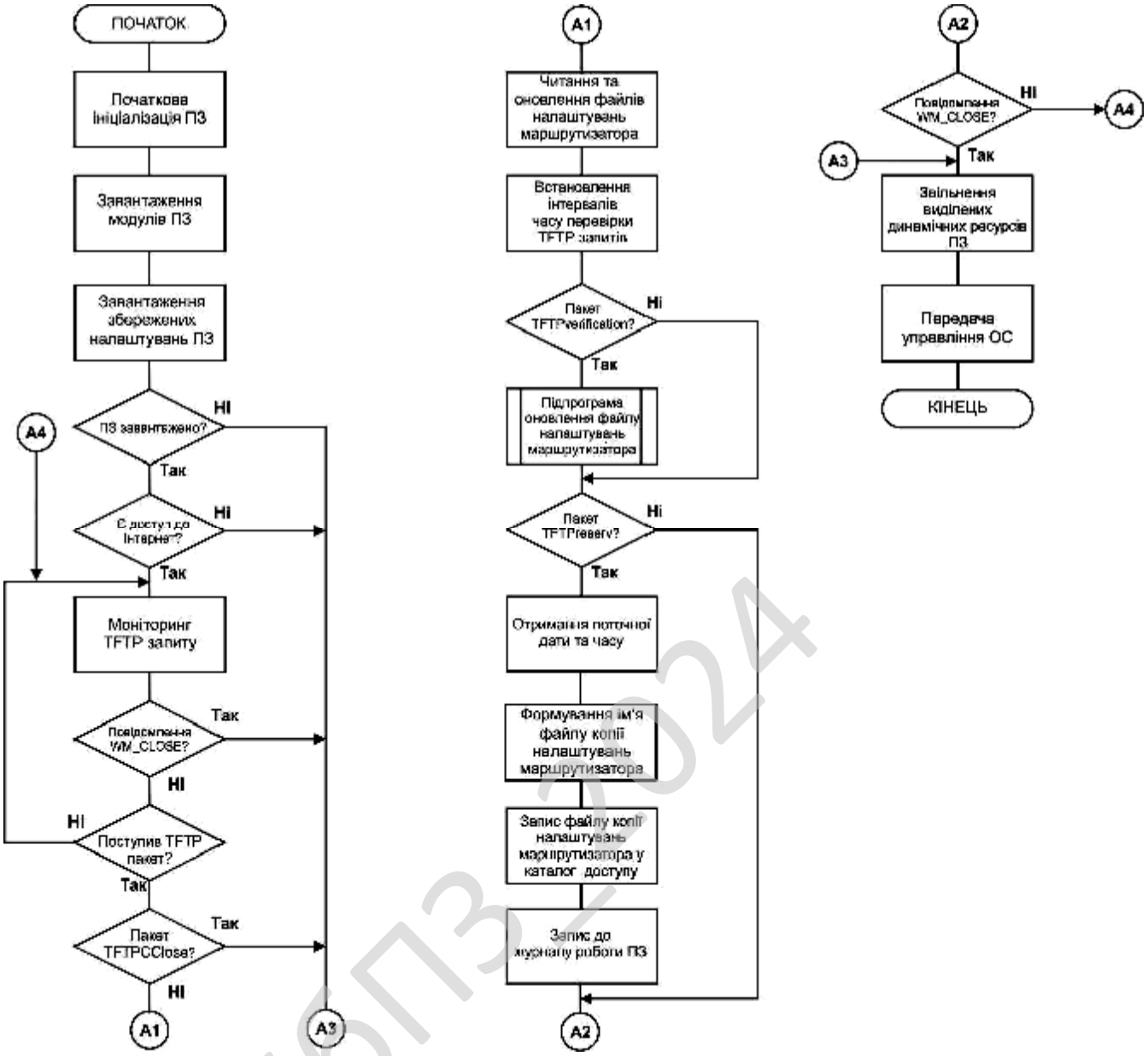

Рисунок 4.1 – Блок-схема основної програми

На рисунку 4.2 зображено роботу підпрограми оновлення файлу налаштувань маршрутизатора. Її робота складається з виконання наступних кроків:

– Моніторинг маршрутиза́торів ЛМ.

– Є доступ до ЛМ (запит).

– Є доступ до маршрутиза́тора (запит).

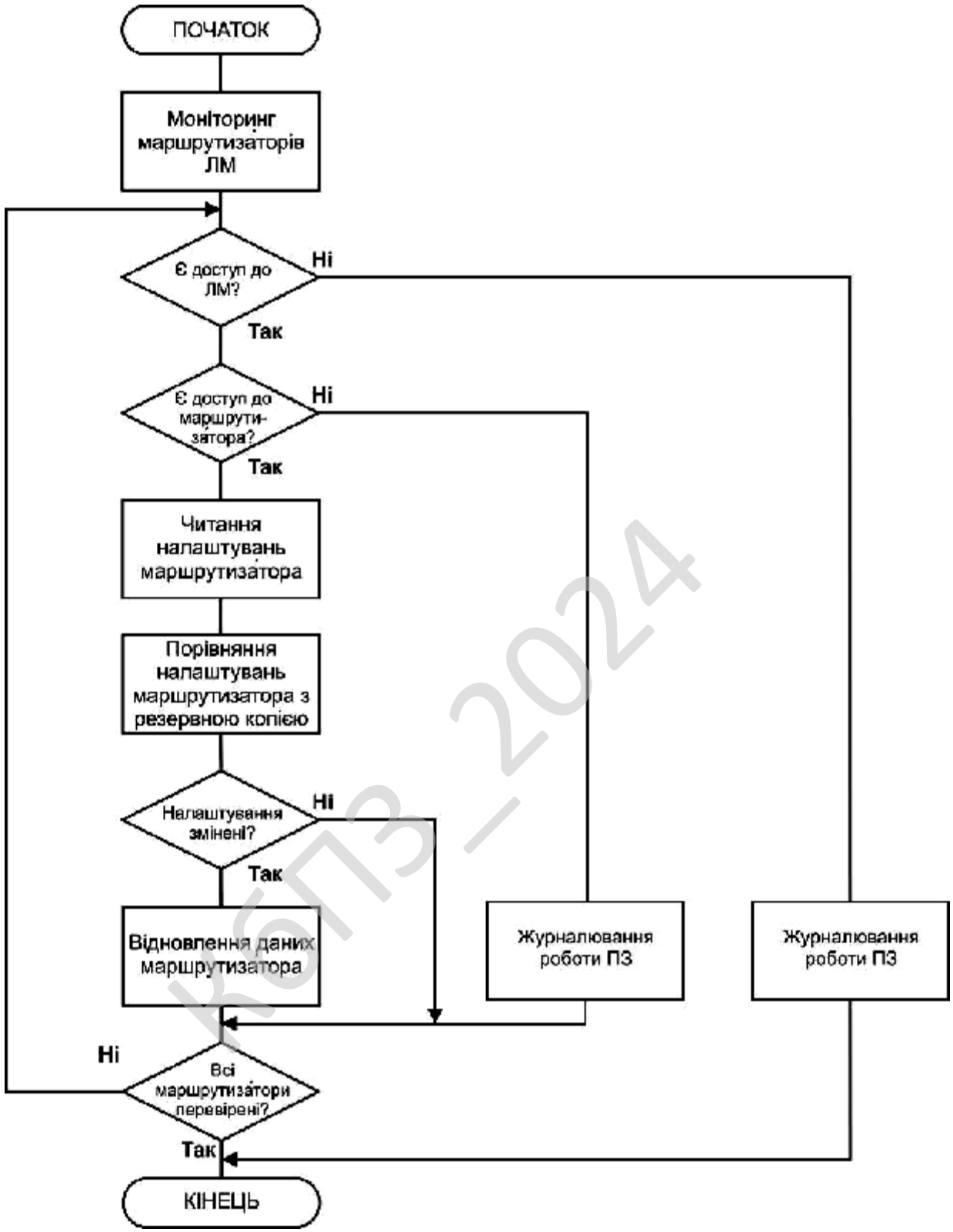

Рисунок 4.2 – Блок-схема підпрограми оновлення файлу налаштувань маршрутизатора

– Читання налаштувань маршрутиза́тора.

– Порівняння налаштувань маршрутизатора з резервною копією.

– Налаштування змінені (запит).

– Відновлення даних маршрутиза́тора.

– Всі маршрутиза́тори перевірені (запит).

Розроблений у бакалаврському проекті програмний пакет, забезпечує універсальну систему захисту за допомогою розповсюдження КК маршрутизаторам.

Метод ПЗ полягає у винесенні модуля генерації секретних ключових послідовностей з розробленого програмного забезпечення в окремий мережний модуль, що дозволяє швидко реагувати на підміну ключів доступу до маршрутизаторів.

Призначення системи це захист маршрутизаторів локальних мереж від деструктивних дій програмних та апаратних помилок.

Захист проводиться інтервальними перевірками налаштувань маршрутизаторів ЛМ, при виявленні помилок проходить автоматичне переналаштування маршрутизатора до нормальних значень через резервне копіювання даних по протоколу Trivial File Transfer Protocol.

Таким чином бакалаврське ПЗ являє собою останній рубіж захисту ЛМ, коли вже проведені деструктивні дії зловмисником чи апаратний збій маршрутизатора блокує ЛМ.

Нижче приведений базовий фрагмент коду який реалізовано у бакалаврському ПЗ, необхідний для однократного посилання ECHO-запиту:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    hIP : THandle;
    pingBuffer : array [0..31] Char;
    pIpe : ^icmp_echo_reply;
    pHostEn : PHostEnt;
    wVersionRequested : WORD;
    lwsaData : WSAData;
    error : DWORD;
```

```
    destAddress : In_Addr;
begin
    // Створюємо handle
    hIP := IcmpCreateFile();
GetMem( pIpe, sizeof(icmp_echo_reply)+ sizeof(pingBuffer));
    pIpe.Data := @pingBuffer;
    pIpe.DataSize := sizeof(pingBuffer);
    wVersionRequested := MakeWord(1,1);
    error := WSAStartup(wVersionRequested,lwsaData);
    if (error <> 0) then
    begin
Memo1.SetTextBuf('Error in call to '+'WSAStartup().');
Memo1.Lines.Add('Error code: '+IntToStr(error));
    Exit;
    end;
        pHostEn := gethostbyname;
    error := GetLastError();
    if (error <> 0) then
    begin
Memo1.SetTextBuf('Error in call to'+'gethostbyname().');
Memo1.Lines.Add('Error code: '+IntToStr(error));
    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(hIPdestAddress.S_addr, pingBuffer sizeof(pingBuffer), Nil, pIpe
sizeof(icmp_echo_reply)+ sizeof(pingBuffer), 5000);
error := GetLastError();
    if (error <> 0) then
    begin
Memo1.SetTextBuf('Error in call to '+'IcmpSendEcho()');
Memo1.Lines.Add('Error code: '+IntToStr(error));
    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(hIP);
    WSACleanup();
    FreeMem(pIpe);
end;
```

Реалізація програмного опитування користувачів. При створенні ПЗ в мережі 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).

За допомогою ICMP хост в мережі обмінюються різною службовою інформацією (інформацією про зміну маршруту, зменшення швидкості передачі, неприступність якої-небудь адреси і т.д.)

В основі протоколу ICMP лежить поняття повідомлень. Повідомлення ICMP протоколу, як правило, оповіщають про помилки, що виникають при обробці датаграмм. ICMP використовує основні властивості протоколу IP, неначебто він був протоколом більш високого рівня. На самій же справі ICMP є складовою частиною IP.

Одним з типів повідомлень протоколу є "відлуння-запиту чи як ще кажуть ехо-запит".Отримавши "ехо-запит" хост зобов'язаний відповісти тому, що послав "ехо-відповіддю".

По суті, "ехо-запит" та "ехо-відповідь» відрізняються лише адресами відправника і одержувача і кодом типу повідомлення (тип 8 – "лунає-запит, тип 0 – "лунає-відповідь).

Реалізації утиліти ping на різних платформах істотно відрізняються. Так, в Ос UNIX використовуються RAW sockets (необроблені, "сирі" сокети), а в ОС Windows всіх версій – ICMP API.

На практиці, у всіх версіях Windows є і використовується бібліотека icmp.dll. Отже, на даний момент, можна її використовувати що було зроблено у бакалаврській роботі.

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

// Час між луна-запитом і луна-відповіддю
        RTTime : u_long;
// в мілісекундах
        DataSize : u_short;
// Розмір повернених даних
        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 посиланих даних
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
//Час очікування відповіді в мілісекундах
) : DWORD; stdcall; external 'ICMP.DLL' name 'IcmpSendEcho';
```

Функція IcmpSendEcho() посилає ICMP запит за заданою IP адресою і поміщає всі відповіді, отримані за час заданого таймауту, в буфер відповідей (ReplyBuffer).

Перед використовуванням функцій Icmp.dll необхідно викликати функцію WSAStartup() з Winsock. Якщо цього не зробити, то перший же виклик функції IcmpSendEcho() приведе до помилки 10091 (WSASYSNOTREADY).

Слід помітити, що відповіді, поміщені в буфер, необов'язково будуть повідомленнями ехо-відповідь. Можливо, що у відповідь на ехо-запит прийдуть повідомлення ICMP про виниклі помилки.

Природно, ці повідомлення так само будуть поміщені в ReplyBuffer. Розмір буфера відповідей повинен бути достатнім для прийому хоча б однієї відповіді, будь то ехо-відповідь або повідомлення про помилку.

Звідси, вказаний розмір складається з розміру самої структури icmp_echo_reply плюс Max(RequestSize, 8), оскільки повідомлення ICMP про помилку складає 8 байт. Функція IcmpSendEcho() повертає кількість відповідей (або структур icmp_echo_reply) в масиві ReplyBuffer.

Якщо функція повернула 0, то для більш детальної діагностики виниклої помилки використовується функція GetLastError().

Для нижче описаного фрагменту коду необхідно:

– створити форму TForm1;

– включити в розділ Uses юніт WinSock;

– помістити на форму компонент Memo1:TMemo і кнопку Button1:TButton;

– в розділі Implementation помістити описи типів і функцій Icmp.dll вище);

– зіставити події OnClick кнопки приведену функцію.

Застосування системи – захисту локальних мереж середніх та великих фірм, підприємств, установ де використовується розгалужена мережа з багатою кількістю маршрутизаторів.

Розроблене бакалаврське ПЗ необхідне адміністраторам для цілодобового моніторингу маршрутизаторів ЛМ.

Кожен маршрутизатор має свої початкові налаштування, при мережних атаках, апаратних, програмних збоях чи невдалих налаштуваннях адміністратора зловмисник може отримати доступ до маршрутизатор та переналаштувати його для власних нужд чи маршрутизатор буде блокувати роботу ЛМ.

Існують погрози для мережі – атака типу розподілена відмова від обслуговування (DDoS, Distributed Denial-of-service).

Бакалаврське ПЗ відсікає цей тип атак. Саме для запобігання цих дій і розроблено бакалаврське програмне забезпечення.

Розподілена відмова від обслуговування – напад на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними до користувачів, для яких комп'ютерна система була призначена.

Для повного представлення розробленого програмного забезпечення, розглянемо головний складальний файл програми.

```
program DiPLOM;
// назва програми
```

```
uses
//модулі, що підключаються
// підключення бібліотек та компонентів та ін.
Forms,
// бібліотека форм
SysUtils, // системні функції
//підключення головного вікна
MAIN in 'MAIN.pas' {Form1},
//підключення вікна налаштувань
frmSettings in 'frmSettings.pas' {Form2},
//підключення вікна списку роутерів
frmROUTER in 'frmROUTER.pas' {Form3},
//підключення вікна журналювання
  frmFIlE in 'frmFIlE.pas' {Form4},
  frmSTAT in 'frmSTAT.pas' {Form5},
//підключення вікна сумуючих даних
  frmSPLASH in 'frmSPLASH.pas' {U_ Splash},
//підключення вікна авторського права
  frmAbout in 'frmAbout.pas' {AboutBox};
//підключення вікна Unit1
frmUnit1 in 'frmUnit.pas' {Unit6};
{$R *.res}
// файл ресурсів
Begin
// Початок
Try
// блок спроби виконання функцій які можуть
// викликати виключення (помилки)
// Створення та виведення на екран вікна заставки ПЗ
    U_Splash:=TU_ Splash.Create(Application);
    U_Splash.Show;
    U_Splash.Update;
    U_Form_Splash.Update;
    U_Splash.Label2.Caption:='чекайте…';
    U_Splash.Update;
    Application.HintHidePause:=7000;
    Application.HintShortPause:=25;
    Application.Initialize;
// Ініціалізація програмного забезпечення
// Створення головного вікна програми
    Application.CreateForm(TForm1, Form1);
// Створення вікна Form2
```

```
        Application.CreateForm(TForm2, Form2);
    // Створення вікна Form3
        Application.CreateForm(TForm3, Form3);
    // Створення вікна Form4
        Application.CreateForm(TForm4, Form4);
    // Створення вікна Form5
        Application.CreateForm(TForm5, Form5);
    // Створення вікна Form6
        Application.CreateForm(Tform6, Form6);
    // Створення вікна AboutBox
        Application.CreateForm(TAboutBox, AboutBox);
    finally
    U_Splash.free;
    //звільнення вікна заставки
    end;
    Application.Run;
    // запуск програмного забезпечення
    end.
```

Реалізація методу обробки запитів маршрутизаторів. Для реалізації задачі обробки запитів від кінцевих маршрутизаторів (для розподілення секретних ключів маршрутизаторам) необхідно використовувати мережні компоненти.

Було використано компонент TNMHTTP (NetMasters HTTP), який знаходиться на вкладці підпрограм які дозволяють реалізувати поставлені задачі:

```
{Для розуміння суті коду стандартний початок модуля не наведено. Йде стандартний заголовок модулю і визначення форми (TForm1) і її екземпляра класу Form1. Також обявлені кнопка TButton та поле TEdit}
    procedure Button1Click(Sender: TObject);
begin
{Намагаємося отримати заголовок}
NMHTTP1.Head(Edit1.Text);
end;
    …
//Викачування вказаної URL в папку розробленого ПЗ
{ПОЧАТОК Йде заголовок файлу і визначення форми TForm1 і її екземпляра Form1. У формі є кнопка TButton і три поля TEdit. При натисненні на кнопку викликається обробник події OnClick – Button1Click. Перед цим в перший TEdit введен адресу URL, в другій – ім'я файлу для заголовка, а в третій – ім'я файлу для тіла html}
procedure Button1Click(Sender: TObject);
begin
{Намагаємося отримати http-документ}
```

```
{Результат треба записати у файли}
    NMHTTP1.InputFileMode := True;
{ вказуємо в які саме файли}
    NMHTTP1.Header := Edit2.Text;
    NMHTTP1.Body := Edit3.Text;
    NMHTTP1.Get(Edit1.Text);
end;
```

Розгляд викачування відразу декількох URL одночасно необхідно розглянути більш детально. У DELPHI дуже легко створювати окремі, підлеглі процеси за допомогою базового класу TThread.

Розглянемо приклад реалізований у бакалаврському програмному забезпеченні – одночасне викачування вказаних URL в заданий каталог.

```
      ….
{... ПОЧАТОК Йде заголовок файлу і визначення форми TForm1 і її екземпляра Form1,
опис класу окремого процесу}
      type
    THTTPThread = class(TThread)
    private
{Для кожного процесу – створюємо свій компонент TNMHTTP}
    FHTTP: TNMHTTP;
    protected
{Execute викликається при запуску процесу; override – замінюємо існуючу процедуру
базового класу TThread}
procedure Execute; override;

{DoWork – створена нами функція, виконання якої синхронізується в Execute}
procedure DoWork;
    public
{URL – створена нами рядок, вказуючий процесу, який URL йому потрібно викачати}
    URL: string;
    end;
{У форму потрібно помістити три кнопки TButton, одне поле TEdit і один список
TListBox. При натисненні на кнопку Button1 викликається обробник події OnClick –
 Button1Click. Перед цим в TEdit потрібно ввести шлях до каталога, в якому
зберігатимуться викачані файли, а ListBox1 потрібно заповнити списком URLлів для
викачування (за допомогою кнопок Add (Button2) і Delete (Button3))}
procedure TForm1.Button3Click(Sender: TObject);
begin
{Видалення виділеного URL із списку}
    if ListBox1.ItemIndex >= 0 then
```

```
        ListBox1.Items.Delete(ListBox1.ItemIndex);
end;

procedure TForm1.Button2Click(Sender: TObject);
    var s: string;
begin
{Додавання URL в список}
    s := InputBox('Input','Введення URL:','');
    if s <> '' then ListBox1.Items.Add(s);
end;

procedure TForm1.Button1Click(Sender: TObject);
  var i: Integer;
begin
{Перевірка на існування каталога}
    if Length(Edit1.Text)> 0 then
        if not DirectoryExists(Edit1.Text) then
            MkDir(Edit1.Text);
{Далі йде створення для кожного URL в списку свого процесу}
    for i := 0 to ListBox1.Items.Count-1 do begin
        with THTTPThread.Create(True) do begin
{Створюємо припинену задачу, указуємо їй її URL і запускаємо її}
            URL := ListBox1.Items[i];
            Resume;
       end;
       end;
       end;
{Оператори процесу THTTPThread}
procedure THTTPThread.Execute;
begin
{Робимо так, щоб кожний процес виконувався одночасно з іншими (синхронізація)}
    Synchronize(DoWork);
end;
       procedure THTTPThread.DoWork;
    var i: Integer;
begin
{Створюємо компонент TNMHTTP}
    FHTTP := TNMHTTP.Create(Form1);
{Результат записуємо у файли}
    FHTTP.InputFileMode := True;
{Підбираємо імена для файлів}
    i:= 1;
```

```
    while FileExists(Form1.Edit1.Text+'\page'+IntToStr(i)+'.htm') do
    Inc(i);
{Вказуємо, в які саме файли класти результати}
    FHTTP.Body := Form1.Edit1.Text+'\body'+IntToStr(i)+'.htm';
    FHTTP.Header := Form1.Edit1.Text+'\header'+IntToStr(i)+'.txt';
{Намагаємося послати запит}
    FHTTP.Get(URL);
    FHTTP.Free;
end;
```

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою Serpent – симетричний блочний алгоритм шифрування, розроблений Россом Андерсоном, Елі Біхамом та Ларсом Кнудсеном. Алгоритм був одним з фіналістів 2-го етапу конкурсу AES. Як і інші алгоритми, які брали участь у конкурсі AES, Serpent має розмір блоку 128 біт і можливі довжини ключа 128, 192 або 256 біт. Алгоритм являє собою 32-раундовий шифр на основі SP-мережі, і працює з блоком з чотирьох 32-бітових слів. Serpent був розроблений так, що всі операції можуть бути виконані паралельно, використовуючи 32-а 1-бітних «потоків».

При розробці Serpent використовувався консервативніший підхід до безпеки, ніж у інших фіналістів AES, проектувальники шифру вважали, що 16 раундів достатньо, щоб протистояти відомим видам криптоаналізу, але збільшили число раундів до 32, щоб алгоритм міг краще протистояти ще не відомим методам криптоаналізу.

Шифр Serpent не запатентований і є громадським надбанням.

Алгоритм створювався під гаслом «криптографічний алгоритм 21 століття» для участі в конкурсі AES. При створенні нового алгоритму Serpent його автори дотримувалися консервативних поглядів на проектування, що підтверджується первісним рішенням про використання таблиць підстановки з відомого багато років раніше алгоритму шифрування DES, який протягом

довгого часу вивчався провідними фахівцями в області криптографії та захисту інформації і чиї властивості і особливості були добре відомі науковому світу. Одночасно з цим до нового алгоритму міг бути застосований вичерпний аналіз, вже розроблений для DES. Не використовувалися нові, неперевірені і невипробувані технології при створенні шифру, який у разі прийняття був би використаний для захисту величезних масивів фінансових транзакцій та урядової інформації. Основною вимогою до учасників конкурсу AES було те, що алгоритм-претендент повинен бути швидшим, ніж 3DES, і надавати як мінімум такий же рівень безпеки: він повинен мати блок даних довжиною 128 біт і ключ завдовжки 256 біт. 16-раундовий Serpent був би таким же надійним, як 3DES, але в два рази швидшим. Однак, автори вирішили, що для більшої надійності варто збільшити кількість раундів до 32. Це зробило шифр таким же швидким, як DES, і набагато надійнішим, ніж 3DES.

Структура алгоритму

Алгоритм Serpent є SP-мережею, у котрій весь блок даних довжиною 128 біт на кожному раунді розбивається на 4 слова довжиною 32 біти. Всі значення, що використовуються при шифруванні є бітовими потоками. Бітові індекси змінюють значення від 0 до 31 для 32-бітових слів, від 0 до 127 – для 128-бітових блоків та від 0 до 255 для 256-бітових ключів тощо. Для внутрішніх обчислень всі біти величин представлені в прямому порядку (little-endian).

Serpent шифрує відкритий текст P довжиною 128 біт в шифротекст C довжиною таких же 128 біт за 32 раунд за допомогою 33 подключів {\displaystyle K_{0},...,K_{32}\,}{\displaystyle K_{0},...,K_{32}\,} довжиною 128 біт. Довжина використовуваного ключа може приймати різні значення, але для конкретики зафіксуємо їх довжину в 128, 192 або 256 біт. Короткі ключі довжиною менше 256 біт доповнюються до повної довжини в 256 біт.

Шифрування складається з наступних основних кроків:

– Початкова перестановка.

– 32 раунд, кожен з яких складається з операції змішування з 128-бітовим ключем (побітове логічне виключаюче «або»), таблична заміна (S-box) і лінійне перетворення. В останньому раунді лінійне перетворення замінюється додатковим накладанням ключа.

– Кінцева перестановка.

Початкова і кінцева перестановки не мають будь-якої криптографічного значущості. Вони використовуються для спрощення оптимізованої реалізації алгоритму і підвищення обчислювальної ефективності.

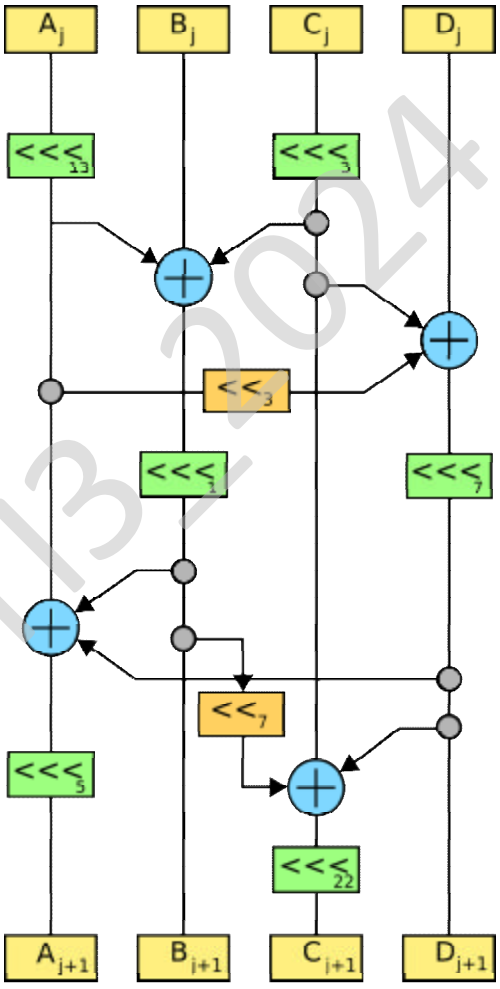

Рисунок 4.3 – Структура алгоритму Serpent

Розширення ключа

Як і інші алгоритми, що брали участь в конкурсі AES, Serpent має можливі довжини ключа 128, 192 або 256 біт. «Неповний» ключ довжиною менше 256 біт доповнюється за наступним правилом: додається одиничний біт справа, за ним слід стільки нульових бітів, щоб довжина ключа стала дорівнює 256 бітам.

Початкова перестановка IP

Дана перестановка IP задається таблицею, де вказується позиція, на яку перейде відповідний біт (наприклад, біт 1 перейде на 32 позицію):

S-бокси (таблиці замін)

В алгоритмі Serpent таблиці замін є 4-бітовими перестановками з властивостями стійкості до диференціального криптоаналізу, до лінійного криптоаналізу і такою властивістю, що порядок вихідних біт, як функції вхідних повинен бути максимальний, тобто бути рівним 3.

Таблиця підстановки генерується з відомих і добре вивчених таблиць для алгоритму DES в ітераційному процесі, поки не будуть отримані бажані диференціальні й лінійні властивості. Таким чином, створюється 8 таблиць підстановки.

Лінійне перетворення LT

Лінійне перетворення LTзадається таблицею, де біти перераховані від 0 до 127 (наприклад, вихідний 2 біт утворений 2, 9, 15, 30, 76, 84, 126 битами, складеними за модулем 2) . В кожному рядку описується 4 вихідних біти, які разом складають вхідні дані на одну таблицю замін в наступному раунді. Варто зазначити, що даний набір являє собою таблицю IP(LT(FP(x))), де LT і є те лінійне перетворення.

Таблиця зворотного лінійного перетворення, яке використовується при розшифровці ILT.

Кінцева перестановка FP

Дана перестановка є зворотною до початкової, тобто $FP=IP^{-1}$ і задається наступною таблицею.

Ефективна реалізація алгоритму

Бажання авторів зробити алгоритм саме таким, яким він є стає зрозумілим при розгляді його ефективної низькорівневої реалізації.

Serpent був створений таким чином, щоб всі операції в процесі шифрування і розшифрування одного блоку могли бути виконані паралельно в 32 потоках. До того ж низькорівневий опис алгоритму набагато простішій, ніж стандартний опис. Ніяких початкових і кінцевих перестановок не потрібно.

Шифрування складається з 32 раундів. Відкритий текст є першими проміжними даними $B_0 = P$. Потім виконується 32 раунди, кожен і-й раунд складається з:

– Змішування з ключем. Проводиться побітове виключаюче «або» проміжних даних B_i з ключем довжиною 128 біт.

– Застосування таблиць підстановки. Вхідні дані довжиною 128 біт поділяються на 4 слова по 32 біта. Таблиця підстановки, реалізована послідовністю логічних операцій (як якщо це було б реалізовано апаратно), застосовується до цих 4 словам. В результаті виходить 4 вихідних слова. Таким чином, центральний процесор виконує підстановку по 32 копій таблиці одночасно.

– Лінійне перетворення. 32-бітові слова перетворюються заданим порядком.

Першою причиною вибору такого лінійного перетворення є максимізація лавинного ефекту. Такі таблиці підстановки мають властивість, що зміна кожного вхідного біта призведе до зміни 2 вихідних бітів. Таким чином, кожен вхідний біт відкритого тексту вже через 3 раунди впливає на всі вихідні біти. Аналогічно кожен біт ключа впливає на результат шифрування.

Друга причина полягає в простоті перетворення. Воно може бути реалізоване на будь-якому сучасному процесорі з мінімальними витратами.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено головне вікно програми. З нього видно, що інтерфейс користувача програми складається з таких логічних блоків як:

– Меню користувача яке складається: Файл; Допомога; Автор розробки.

– Лівого блоку встановлення часу перевірки.

– Правих функціональних кнопок: Сканування; Налаштування ПЗ; Журнал роботи ПЗ; Встановлення часу перевірки; Додати; Видалити.

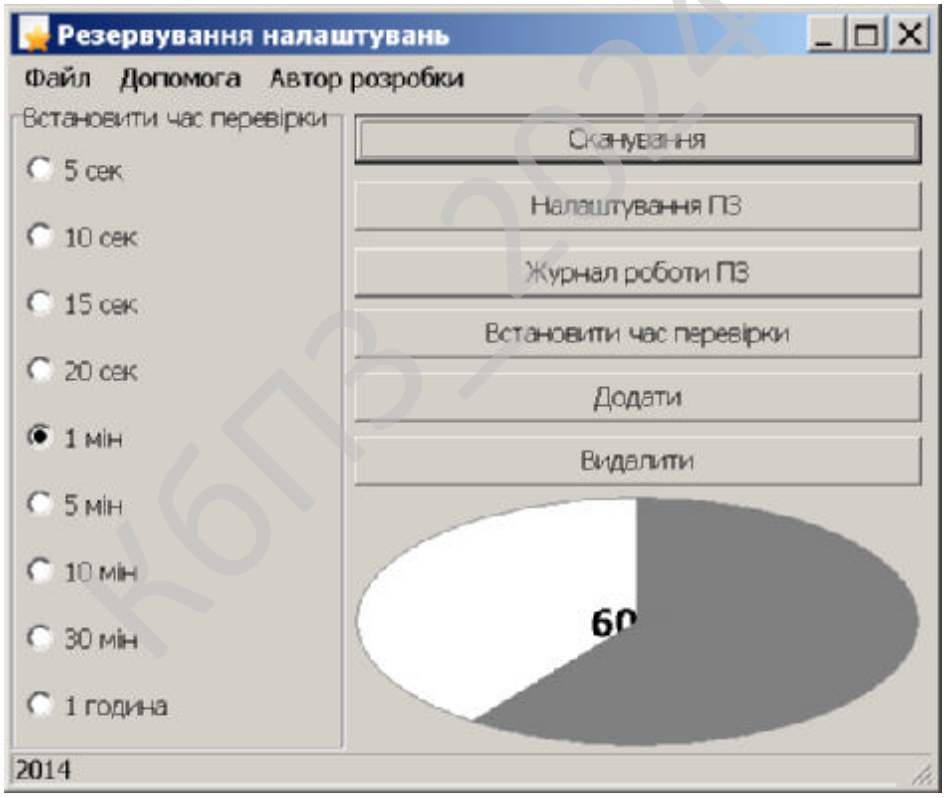

Рисунок 5.1 – Головне вікно програми

На рисунку 5.2 зображено форму авторського права. Обрано умови розповсюдження – Shareware. Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на

ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (неповнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми. В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання. Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (дискету чи CD-ROM). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

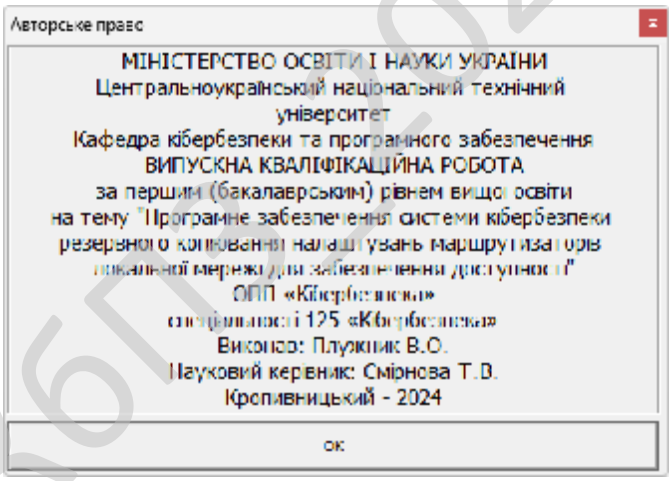

Рисунок 5.2 – Довідка автора

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

– Досліджена система резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

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 RAD Studio Delphi. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для

системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності. 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 кібербезпеки й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Serpent.

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. Оліфер В.Г. Комп'ютерні мережі. Принципи, технології, протоколи. Підручник / В.Г. Оліфер, Н.А.Оліфер. – [5-е вид.]. – 2016. – 944 с.

2. Е. Таненбаум, Д. Уезеролл «Комп'ютерні мережі». – [5-е вид.]. – 2016. – 960 с.

3. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 1». Cisco Press. 2020. – 848 p.

4. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 2 Premium Edition eBook and Practice Test». Cisco Press. 2020. – 624 p.

5. Scott Jernigan «CompTIA Network+ Certification All-in-One Exam Guide, Eighth Edition». 2022. – 976 p.

6. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.

7. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

8. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

9. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

10. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

11. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

12. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». *Lecture Notes on Data Engineering and Communications Technologies*, 2023, 178, pp. 208–223.

13. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

14. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». *Communications in Computer and Information Science*, 2021, vol 1486. Springer, Cham. pp 169-184.

15. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

16. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

17. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

18. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

19. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

20. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379.

21. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645.

22. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». *International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019*; Odessa; Ukraine; 9-13 September 2019. P.22-28.

23. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

24. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

25. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

26. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

27. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

28. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

29. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

30. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

31. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

32. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

33. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду

абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». *Сучасні інформаційні системи,* 2023, том 7, № 2, С. 49-56.

34. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

35. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

36. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

37. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

38. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

39. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» *Вісник Черкаського державного технологічного університету. Технічні науки.* №4. С. 103-110. 2020.

40. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020.

41. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

42. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

43. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

44. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

45. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

46. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139

47. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для

моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

48. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

49. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

50. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.

51. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

52. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Алгоритми формування безлічі маршрутів передачі метаданих у антивірусні хмарні системи. Збірник наукових праць "Системи обробки інформації". - Випуск 5 (142). - Х.: ХУПС - 2016. - С. 148-152.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 136-02 від 01.04.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти 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 кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності;

– 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, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

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 10/11.

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 RAD Studio Delphi.

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 – 81 аркуш.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

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.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2024 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 8.06.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Смірнова Т.В.

Програмне забезпечення системи кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі для забезпечення доступності

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 34

Літера: РП

Кропивницький – 2024 року

ОСНОВНИЙ ФАЙЛ БАКАЛАВРСЬКОГО ПЗ – PROJECT1.DPR

```
{ Авторське право ПЗ
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Реалізація бакалаврського проекту на тему – програмне забезпечення системи
кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі
для забезпечення доступності
Розробник: Плужник Владислав Олександрович
2024 рік, гр. КБ-21-3СК
}

program Project1; // назва програми
uses //модулі, що підключаються
  // підключення бібліотек та компонентів та ін.
  Forms, // основна бібліотека форм VCL
  SysUtils, // системні функції VCL
  M in 'M.pas' {Form1}, //підключення головного вікна
  S in 'S.pas' {Form2}, //підключення вікна налаштувань
  R in 'R.pas' {Form3}, //підключення вікна
  L in 'L.pas' {Form4}, //підключення вікна логу роботи ПЗ
  K in 'K.pas' {Form5}, //підключення вікна KEYSTAT
  SPL in 'SPL.pas' {U_SPL},
  A in 'A.pas' {AboutBox}; //підключення вікна авторського права
  U1 in 'U1.pas' {Unit6};//підключення вікна модуля

{$R *.res} // файл стандартних ресурсів

Begin // Початок програми
try
    // Створення та виведення на екран вікна заставки ПЗ
    U_SPL:=TU_SPL.Create(Application);
    U_SPL.Show; // Показ
    U_SPL.Update; // Оновлення вікна
    U_SPL.Label2.Caption:='Завантаження';
    U_SPL.Update; // Оновлення вікна

  Application.HintShortPause:=25;
  Application.Initialize; // Ініціалізація програмного забезпечення
  Application.CreateForm(TForm1, Form1); // Створення головного вікна програми
  Application.CreateForm(TForm2, Form2); // Створення вікна Form2
  Application.CreateForm(TForm3, Form3); // Створення вікна Form3
  Application.CreateForm(TForm4, Form4); // Створення вікна Form4
  Application.CreateForm(TForm5, Form5); // Створення вікна Form5
  Application.CreateForm(Tform6, Form6); // Створення вікна Form6
  Application.CreateForm(TAboutBox, AboutBox); // Створення вікна AboutBox
  Finally
    U_SPL.free; //звільнення вікна заставки
end;
  Application.Run; // запуск програмного забезпечення
end.
```

МОДУЛЬ НАЛАШТУВАННЯ МАРШРУТИЗАТОРІВ

```
{ Авторське право ПЗ
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Реалізація бакалаврського проекту на тему – програмне забезпечення системи
кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі
для забезпечення доступності
Розробник: Плужник Владислав Олександрович
2024 рік, гр. КБ-21-3СК
}

unit SET_DATA; // Назва модулю налаштування маршрутизаторів

interface //Інтерфейсна частина (об’ява функцій)

uses // Підключення бібліотек
   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 //створення нового класу TIpConnInfo
   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;

// об'ява класу EConnListError типу виключення (обробник помилок)
EConnListError = class(Exception);
   EConnListLockError = class(EConnListError);
   EConnListUnlockError = class(EConnListError);
   EConnListRefreshError = class(EConnListError);

// структура статусів
TIpConnListStatus = ( connlist_ready, connlist_refreshing );
TIpProtocol = (udp_ip, tcp_ip);
TIpProtocols = set of TIpProtocol;
// об'ява класу TFnugryIpConnectionList
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;

// об'ява класу типу виключення (обробник помилок)
ENetstatError = class(Exception);
TNetstatCounterObject = class //створення нового класу TNetstatCounterObject
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) //створення нового класу TCounterList
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) //створення нового класу TIpStats
   public
      constructor Create( aOwner :TComponent ); override;
   end;

   TIcmpStats = class(TCounterList) //створення нового класу TIcmpStats
   public
      constructor Create( aOwner :TComponent ); override;
   end;

   TTCPStats = class(TCounterList) //створення нового класу TTCPStats
   public
      constructor Create( aOwner :TComponent ); override;
   end;

   TUdpStats = class(TCounterList) //створення нового класу TUdpStats
   public
      constructor Create( aOwner :TComponent ); override;
   end;

implementation // реалізація модулю (вихідний код)

const // об'ява констант

  DEFAULT_LOCK_TIMEOUT  = 100; // час у мс
  SNMP_LIB_NAME         = 'inetmib1.dll'; // використовуваєма бібліотека
  SNMP_INITPROC_NAME    = 'SnmpExtensionInit'; // Початкова функція
  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 }

// реалізація конструктору класу TIpStats
constructor TIpStats.Create( aOwner :TComponent );
begin
   inherited;
   AddCounter( TNetstatCounterObject.Create( 'ipDefaultTTL', '',
mib_ipDefaultTTL));
```

```
   AddCounter( TNetstatCounterObject.Create( 'ipInReceives', '',
mib_ipInReceives));
   AddCounter( TNetstatCounterObject.Create( 'ipInHdrErrors', '',
mib_ipInHdrErrors));
end;

// Реалізація методу класу
{ TIcmpStats }
// Реалізація методу класу
{ TNetstatCounterObject }
function TNetstatCounterObject.GetValue :DWORD;
var // опис змінних корисувача
   NET     :TRFC1157VarBind;
   NETlist :TRFC1157VarBindList;
   errorStatus :TAsnInteger;
   errorIndex  :TAsnInteger;
begin
   NETlist.List := @NET;
   NETlist.len := 1;
   fillchar(NET, SizeOf(NET), 0);
   NET.Name.idLength := mibLen;
   NET.Name.ids := @FId;
   if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
      NETlist, errorStatus, errorIndex) then
      raise ENetstatError.Create(m_err_query);
   if not (NETlist.list.value.asnType in
     [ASN_GAUGE32, ASN_INTEGER, ASN_INTEGER32, ASN_COUNTER32, ASN_UNSIGNED32])
then
      raise ENetstatError.Create(m_err_invtype);
   result := NETlist.list.value.Counter;
end;
// реалізація конструктору класу TNetstatCounterObject
constructor TNetstatCounterObject.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, 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;

// деструктор класу TNetstatCounterObject
destructor TNetstatCounterObject.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;
// реалізація конструктору класу TCounterList
constructor TCounterList.Create( aOwner :TComponent );
begin
   inherited;
   FCounters := TStringList.Create;
end;

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

```
end;

procedure TCounterList.Clear;
var // опис змінних корисувача
   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 }

// конструктор 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;

// реалізація функції отримання портів TCP/IP локальної машини
function TIpConNInfo.GetLocalPort :Integer;
begin
   result := FLocalPort;
end;

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

// реалізація функції отримання відключених портів TCP/IP
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 }

// реалізація функції отримання кількості маршр.
function TFnugryIpConnectionList.GetConnCount :Integer;
begin
   assert(assigned(FConnections));
   result := FConnections.Count;
end;

// реалізація функції підключеня до маршрутизатора
function TFnugryIpConnectionList.GetConnections( Index :Integer ):TIpConnInfo;
begin
   assert(assigned(FConnections));
   result := FConnections[Index];
end;

// реалізація процедури очищення списку пК у мережі
procedure TFnugryIpConnectionList.Clear;
var // опис змінних корисувача
   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;

// конструктор класу TFnugryIpConnectionList
constructor TFnugryIpConnectionList.Create( aOwner :TComponent );
var // опис змінних корисувача
   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, 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;

// деструктор класу TFnugryIpConnectionList
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;

// процедура отримання інформації ПК у ЛМ
procedure TFnugryIpConnectionList.Refresh;
   procedure ReadTcpTable; // реалізація процедури читання TCP таблиці
   var // опис змінних корисувача
      NET     :TRFC1157VarBind;
      NETlist :TRFC1157VarBindList;
      errorStatus :TAsnInteger;
      errorIndex  :TAsnInteger;
      ConnIndex   :Integer;
      Info        :TIpConnInfo;
      ListTail    :Integer;
   begin
      fillchar( NETlist, SizeOf(NETlist), 0);
      NETlist.List := @NET;
      NETlist.len := 1;
      fillchar(NET, SizeOf(NET), 0);
      NET.Name.idLength := mibLen;
      NET.name.ids := @mib_tcpConnTable;
      ListTail    := FConnections.Count;
      if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
         NETlist, errorStatus, errorIndex) then
```

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

```
    while (NETlist.list.value.asnType = ASN_INTEGER) do
    begin
       Connections[ConnIndex].FRemotePort := NETlist.list.value.counter;
       if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
          NETlist, errorStatus, errorIndex) then
          raise EConnListRefreshError.Create(m_err_query);
       inc(ConnIndex);
    end;
end;

// реалізація процедури читання Udp таблиці
procedure ReadUdpTable;
 var // опис змінних корисувача
    NET     :TRFC1157VarBind;
    NETlist :TRFC1157VarBindList;
    errorStatus :TAsnInteger;
    errorIndex  :TAsnInteger;
    ConnIndex   :Integer;
    Info        :TIpConnInfo;
    ListTail    :Integer;
 begin
    fillchar( NETlist, SizeOf(NETlist), 0);
    NETlist.List := @NET;
    NETlist.len := 1;
    fillchar(NET, SizeOf(NET), 0);
    NET.Name.idLength := mibLen;
    NET.name.ids := @mib_udpTable;
    ListTail := FConnections.Count;
    if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
       NETlist, errorStatus, errorIndex) then
       raise EConnListRefreshError.Create(m_err_query);
    if NETlist.list.value.asnType = ASN_NULL then
       exit;
   { читання локального порту }
    while (NETlist.list.value.asnType = ASN_RFC1155_IPADDRESS) do
    begin
       Info := TIpConnInfo.Create('UDP');
       move(NETlist.list.value.address.stream^,
          Info.FLocalIP,
          sizeof(TInAddr));
       FConnections.Add(Info);
       if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
          NETlist, errorStatus, errorIndex) then
          raise EConnListRefreshError.Create(m_err_query);
    end;
    if NETlist.list.value.asnType = ASN_NULL then
       raise EConnListRefreshError.Create(m_err_queryend);
    { читання локального порту }
    ConnIndex := ListTail;
    while (NETlist.list.value.asnType = ASN_INTEGER) do
    begin
       Connections[ConnIndex].FLocalPort := NETlist.list.value.counter;
       if not TSnmpQueryProc(FSnmpQueryProc)(ASN_RFC1157_GETNEXTREQUEST,
```

```
            NETlist, 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.
```

КЛЮЧІ КОМПІЛЯТОРА DATA.DOF

```
[Compiler] {Налаштування компілятора для компіляції розробленого ПЗ}
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=Laskovskiy Inc
FileDescription=
FileVersion=1.0.0.0
InternalName=
LegalCopyright=
LegalTrademarks=
OriginalFilename=
ProductName=2014 year
ProductVersion=2.5.0.0
Comments=Diplom PZ
```

МОДУЛЬ НАЛАШТУВАНЬ ЛМ

```
unit B_Net; // Назва модулю налаштування

interface //Інтерфейсна частина (об'ява функцій)
// Підключення бібліотек
uses Windows, ShellAPI, ShlObj, ActiveX, ComObj, Dim, Classes, SysUtils,
WinSock;

type // Опис типів даних та класів
  //Опис класів виключення (обробник помилок)
  ComputerFound = class (Exception);
  ECannotFindNetwork = class (Exception);

  //створення нового класу TROUTERStringObject
  TROUTERStringObject = class (TObject)
  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;

  //створення нового класу TROUTERStringObjectArray
  TROUTERStringObjectArray = class (TDynamicArray)
  private
    function GetObject(Index: Integer): TROUTERStringObject;
    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: TROUTERStringObject): 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;

  //створення нового класу TROUTERStringObjectList
  TROUTERStringObjectList = class (TStrings)
  private
    FArray: TROUTERStringObjectArray;
    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;

{TNetworkWorkgroup - 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}

// об’ява класу TNetworkWorkgroup від TROUTERStringObjectList
 TNetworkWorkgroup = class (TROUTERStringObjectList);

//створення нового класу TNetworkNeighborhood
 TNetworkNeighborhood = class (TROUTERStringObjectList)
  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;
    function EnumObjects(ShellFolder: IShellFolder): IEnumIDList;
    procedure ParseFolder(Folder: IShellFolder; Items:
                  TROUTERStringObjectList; StorePIDLs: Boolean = False);
    procedure ParseFolderEx(Folder: IShellFolder; Items: TStrings);
    function FreeRefObj(Index: Integer; var Obj: TROUTERStringObject): Integer;
    function GetWorkgroup(Name: TString): TNetworkWorkgroup;
  public
    procedure Refresh;
    property Workgroup[Name: TString]: TNetworkWorkgroup 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;

// об'ява процедур і функцій
function GetIPAddress(NetworkName: TString): TString;
procedure GetIPAddresses(Network: TNetworkNeighborhood; List: TStrings);
function EnumSharedResources(ComputerName: TString; List: TStrings): Boolean;


implementation // реалізація модулю (вихідний код)

uses DimConst; // Підключення бібліотек

// Реалізація методу класу
{ TROUTERStringObject }

procedure TROUTERStringObject.SetData(const Value: Pointer);
begin
  FData := Value;
end;

procedure TROUTERStringObject.SetRefObj(const Value: TObject);
begin
  FRefObj := Value;
end;

procedure TROUTERStringObject.SetTag(const Value: Integer);
begin
  FTag := Value;
end;
```

```
procedure TROUTERStringObject.SetValue(const Value: TString);
begin
  FValue := Value;
end;

// Реалізація методу класу
{ TROUTERStringObjectArray }
function TROUTERStringObjectArray.Add: Integer;
begin
 Result:=inherited Add;
 CreateItem(Result);
end;

function TROUTERStringObjectArray.AddItem(const Item): Integer;
begin
 Result:=Add;
end;

// реалізація конструктору класу TROUTERStringObjectArray
constructor TROUTERStringObjectArray.Create;
begin
 inherited Create(0, SizeOf(TROUTERStringObject));
end;

procedure TROUTERStringObjectArray.CreateItem(Index: Integer);
var // опис змінних користувача
 P: ^TROUTERStringObject; // створення вказівника
begin
 P:=GetItemPtr(Index);
 P^:=TROUTERStringObject.Create;
end;

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

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

// деструктор класу TROUTERStringObjectArray
destructor TROUTERStringObjectArray.Destroy;
begin
 ForEach(Integer(Self), @TROUTERStringObjectArray.FreeObject);
 inherited;
end;

procedure TROUTERStringObjectArray.FreeItem(Index: Integer);
var // опис змінних користувача
```

```
 P: ^TROUTERStringObject;
begin
 P:=GetItemPtr(Index);
 FreeAndNil(P^);
end;

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

function TROUTERStringObjectArray.GetData(Index: Integer): Pointer;
begin
 Result:=GetObject(Index).Data;
end;

function TROUTERStringObjectArray.GetObject(Index: Integer):
TROUTERStringObject;
begin
 GetItem(Index, Result);
end;

function TROUTERStringObjectArray.GetRefObj(Index: Integer): TObject;
begin
 Result:=GetObject(Index).RefObj;
end;

function TROUTERStringObjectArray.GetTag(Index: Integer): Integer;
begin
 Result:=GetObject(Index).Tag;
end;
// реалізація функції отримання значення
function TROUTERStringObjectArray.GetValue(Index: Integer): TString;
begin
 Result:=GetObject(Index).Value;
end;

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

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

procedure TROUTERStringObjectArray.SetCount(const NewCount: Cardinal);
var // опис змінних корисувача
 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;
// реалізація функції встановлення значення класу TROUTERStringObjectArray
procedure TROUTERStringObjectArray.SetData(Index: Integer; const Value:
Pointer);
begin
 GetObject(Index).Data:=Value;
end;

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

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

procedure TROUTERStringObjectArray.SetValue(Index: Integer; const Value:
TString);
begin
 GetObject(Index).Value:=Value;
end;

// Реалізація методу класу
{ TROUTERStringObjectList }

function TROUTERStringObjectList.Add(const S: string): Integer;
begin
 Result:=FArray.Add;
 FArray.Value[Result]:=S;
end;

procedure TROUTERStringObjectList.Clear;
begin
 FArray.Count:=0;
end;
// реалізація конструктору класу TROUTERStringObjectList
constructor TROUTERStringObjectList.Create;
begin
 inherited Create;
 FArray:=TROUTERStringObjectArray.Create;
end;
```

```
procedure TROUTERStringObjectList.Delete(Index: Integer);
begin
 FArray.Delete(Index);
end;

// деструктор класу TROUTERStringObjectList
destructor TROUTERStringObjectList.Destroy;
begin
 FArray.Free;
 inherited;
end;

procedure TROUTERStringObjectList.Exchange(Index1, Index2: Integer);
begin
 FArray.Swap(Index1, Index2);
end;

function TROUTERStringObjectList.Get(Index: Integer): string;
begin
 Result:=FArray.Value[Index];
end;

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

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

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

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

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

procedure TROUTERStringObjectList.Put(Index: Integer; const S: string);
begin
 FArray.Value[Index]:=S;
end;
```

```
procedure TROUTERStringObjectList.PutObject(Index: Integer; AObject: TObject);
begin
 FArray.RefObj[Index]:=AObject;
end;

procedure TROUTERStringObjectList.SetData(Index: Integer; const Value: Pointer);
begin
 FArray.Data[Index]:=Value;
end;

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

// Реалізація методу класу
{ TNetworkNeighborhood }

function TNetworkNeighborhood.Add(const S: string): Integer;
begin
 Result:=inherited Add(S);
 Objects[Result]:=TNetworkWorkgroup.Create;
end;

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

function TNetworkNeighborhood.CopyPIDL(IDList: PItemIDList): PItemIDList;
var // опис змінних корисувача
 Size: Integer;
begin
 Size := GetPIDLSize(IDList);
 Result := CreatePIDL(Size);
 if Assigned(Result) then CopyMemory(Result, IDList, Size);
end;

// реалізація конструктору класу TNetworkNeighborhood
constructor TNetworkNeighborhood.Create;
begin
 inherited Create;
 Refresh;
end;

function TNetworkNeighborhood.CreatePIDL(Size: Integer): PItemIDList;
var // опис змінних корисувача
 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 TNetworkNeighborhood.Delete(Index: Integer);
begin
end;

procedure TNetworkNeighborhood.DisposePIDL(ID: PItemIDList);
var // опис змінних корисувача
 Malloc: IMalloc;
begin
 if ID = nil then Exit;
 OLECheck(SHGetMalloc(Malloc));
 Malloc.Free(ID);
end;

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

function TNetworkNeighborhood.FindComputer(Name: TString): TString;
var // опис змінних корисувача
 i, j: Integer;
 List: TNetworkWorkgroup;
 S: TString;
begin
 Result:='';
 try
  for i:=0 to Count - 1 do begin
   List:=Objects[i] as TNetworkWorkgroup;
   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 TNetworkNeighborhood.FreeRefObj(Index: Integer;
  var Obj: TROUTERStringObject): Integer;
begin
 FreeAndNil(Obj.FRefObj);
 Result:=0;
end;

function TNetworkNeighborhood.GetDisplayName(ShellFolder: IShellFolder;
  PIDL: PItemIDList): TString;
var // опис змінних корисувача
 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 TNetworkNeighborhood.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 TNetworkNeighborhood.GetPrevPIDL(PIDL: PItemIDList): PItemIDList;
var // опис змінних корисувача
 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 TNetworkNeighborhood.GetWorkgroup(Name: TString): TNetworkWorkgroup;
var // опис змінних корисувача
 Index: Integer;
```

```
begin
 Index:=IndexOf(Name);
 if Index<>-1 then Result:=Objects[Index] as TNetworkWorkgroup else Result:=nil;
end;

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

// реалізація процедури отримання списку ПК
procedure TNetworkNeighborhood.ListComputers(Strings: TStrings);
var // опис змінних корисувача
 i, j: integer;
 L: TNetworkWorkgroup;
 S: TString;
begin
 Strings.BeginUpdate;
 try
  Strings.Clear;
  for i:=0 to Count - 1 do begin
   L:=Objects[i] as TNetworkWorkgroup;
   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 TNetworkNeighborhood.ListNetwork(Strings: TStrings);
var // опис змінних корисувача
 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 TNetworkNeighborhood.NextPIDL(IDList: PItemIDList): PItemIDList;
begin
 Result := IDList;
 Inc(PChar(Result), IDList^.mkid.cb);
end;

function TNetworkNeighborhood.OriginFolder: IShellFolder;
var  // опис змінних корисувача
 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 TNetworkNeighborhood.OriginFolderNT:IShellFolder;
var // опис змінних корисувача
 Desktop: IShellFolder;
 S: TString; W: WideString; P: PWideChar;
 Len, Flags: LongWord;
 Machine, Workgroup, Network: PItemIDList;
 NetShell: IShellFolder;
 Enum: IEnumIDList;
 ID: PItemIDList;
begin
 S:='\\'+GetComputerName; //отримання ім'я ПК ЛМ
 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;

procedure TNetworkNeighborhood.ParseFolder(Folder: IShellFolder;
  Items: TROUTERStringObjectList; StorePIDLs: Boolean);
var // опис змінних корисувача
 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;

procedure TNetworkNeighborhood.ParseFolderEx(Folder: IShellFolder;
  Items: TStrings);
var // опис змінних корисувача
 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 TNetworkNeighborhood.Refresh;
var // опис змінних корисувача
 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 TROUTERStringObjectList, False);
   Workgroup:=nil;
  end;
 except
  raise ECannotFindNetwork.Create(SCannotFindNetwork);
 end;
end;

procedure TNetworkNeighborhood.StripLastID(IDList: PItemIDList);
var // опис змінних корисувача
 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 // опис змінних корисувача
 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  // опис змінних корисувача
 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 // опис змінних корисувача
 ShellFolder: IShellFolder;
begin
 ShellFolder:=GetShellFolder(ComputerName);
 Result:=Assigned(ShellFolder);
 if Result then TNetworkNeighborhood.ParseFolderEx(ShellFolder, List);
end;

function GetIPAddress(NetworkName: TString): TString;
var // опис змінних корисувача
 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 АВТОРСЬКОГО ПРАВА

```
// Модуль авторського права
unit About; // Назва модулю

// авторське право ПЗ
{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Факультет механіко-технологічний
Кафедра кібербезпеки та програмного забезпечення
Реалізація бакалаврського проекту на тему – програмне забезпечення системи
кібербезпеки резервного копіювання налаштувань маршрутизаторів локальної мережі
для забезпечення доступності
Розробник: Плужник Владислав Олександрович
2024 рік, гр. КБ-21-3СК
}
Interface //Інтерфейсна частина (об’ява функцій)
// Підключення бібліотек
uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls, StdCtrls,
     Buttons, ExtCtrls;

  // Опис типів даних та класів
  type
    TAboutBox = class(TForm) // об’ява класу TAboutBox
      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 // опис змінних корисувача
    AboutBox: TAboutBox;
  Implementation // реалізація модулю (вихідний код)

  {$R *.DFM} //ресурси модуля

  // Розроблена функція ПЗ
  procedure TAboutBox.OKButtonClick(Sender: TObject);
  begin //початок
   close;
  end; //кінець

  end.
```