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 другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-23М
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Гречушкін С.В.
«____» _____ 2024 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Коваленко А.С.
«____» _____ 2024 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *магістр*
Галузь знань _12 _____*"Інформаційні технології"*_____
Спеціальність 122 *"Комп'ютерні науки"*
Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерні науки"*_

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

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Гречушкіну Сергію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data*

2. Керівник роботи *Коваленко Анна Степанівна, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 18-13 від 07.08.2024 року

3. Строк подання студентом роботи до захисту *2.12.2024 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data*

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

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Маркетингове та економічне обґрунтування ІТ-проєкту.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Доренська А.О.	05.10.2024	14.11.2024
Охорона праці	Марченко К.М., к.т.н., доцент	06.10.2024	16.11.2024

7. Дата видачі завдання « 6 » вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2024 р.	
3.	Розробка моделі компонента	20.10.2024 р.	
4.	Розробка структур даних	25.10.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2024 р.	
6.	Програмування алгоритмів	10.11.2024 р.	
7.	Розрахунок економічної ефективності	13.11.2024 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2024 р.	
9.	Оформлення ПЗ	17.11.2024 р.	
10.	Попередній захист роботи	2.12.2024 р.	

Дата видачі завдання
« 6 » вересня 2024 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2024 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Гречушкін С.В. Дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Метою розробки є дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Об'єктом дослідження є процес оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Предметом дослідження є методи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Методи дослідження базуються на методах Big Data та теорії надійності, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

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 Visual C#.

Ключові слова: Комп'ютерні науки, оптимізація, продуктивність, надійність, мережні системи зберігання, Big Data

ABSTRACT

Grechushkin S.V. Research and software implementation of a system for optimizing the performance and reliability of network storage systems Big Data. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this final qualification work for the second (master's) level of higher education, software has been developed, which is intended for a system for optimizing the performance and reliability of network storage systems Big Data.

The purpose of the development is the research and software implementation of a system for optimizing the performance and reliability of network storage systems Big Data.

The object of the research is the process of optimizing the performance and reliability of network storage systems Big Data.

The subject of the research is methods for optimizing the performance and reliability of network storage systems Big Data.

The research methods are based on Big Data methods and reliability theory, methods of mathematical statistics, and methods of software development.

The result of the work is a software implementation of a system for optimizing the performance and reliability of network storage systems Big Data.

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 the software are provided.

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

The program was developed in the Visual C# environment.

Keywords: Computer science, optimization, performance, reliability, network storage systems, Big Data

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АВІ	–	адміністратор віртуальної інфраструктури
АІБ	–	адміністратор інформаційної безпеки
ВІ	–	віртуальна інфраструктура
ЗЗІ	–	засоби захисту інформації
ПД	–	персональні дані
ПЗ	–	програмне забезпечення
ЦЗОД	–	центр зберігання й обробки даних
ЦОД	–	центр обробки даних
DLP	–	захист від витоків даних
IPS	–	системи запобігання вторгнень
CIRC	–	Cross Interleaved Reed Solomon Code
EAB	–	Embedded Array Block, блок зосередженої пам'яті
ECC	–	error-correcting code, код корекції помилок
FEC	–	метод прямої корекції помилок
LDPC	–	коди Галлагера
NAK	–	негативне підтвердження

ВСТУП

Актуальність теми. Централізоване зберігання даних для систем зберігання Big Data на підключені до мережі ресурсах має безліч переваг – як для малого бізнесу, так і для корпоративних ЦОД. У той же час при централізованому зберіганні можливі проблеми продуктивності – у випадку інтенсивного доступу до тих або інших ресурсів або при виконанні ресурсномістких додатків. Як же забезпечити достатню продуктивність при виборі мережної інфраструктури для зберігання даних і при налаштуванні її конфігурації з урахуванням потреб організації?

Незалежно від того, чи використовуєте ви мережа зберігання даних SAN або пристрій NAS, швидше за все, ті або інші критичні корпоративні дані зберігаються в якійсь мережній файловій системі або, у всякому разі, повинні в ній зберігатися. Їхнє спільне використання означає більше ефективний розподіл ємності зберігання, простий доступ до корпоративної інформації й захищеність за допомогою таких засобів, як RAID. Централізоване зберігання означає також більше просте керування ресурсами зберігання, а запланованe створення резервних копій простіше виконувати у випадку мережних систем зберігання, чим при використанні DAS.

У той же час при централізованому зберіганні можливі проблеми продуктивності – у випадку інтенсивного доступу до тих або інших ресурсів або при виконанні ресурсномістких додатків. Як же забезпечити достатню продуктивність при виборі мережної інфраструктури для зберігання даних і налаштуванні її конфігурації для потреб організації?

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

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

– Огляд існуючих систем оптимізації продуктивності та надійності мережних систем зберігання Big Data.

– Дослідження системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

– Програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Об'єктом дослідження є процес оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Предметом дослідження є методи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Методи дослідження базуються на методах Big Data та теорії надійності, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод оптимізації продуктивності та надійності мережних систем зберігання Big Data.

– Розроблено вітчизняний продукт оптимізації продуктивності та надійності мережних систем зберігання Big Data, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти LV науково-технічної конференції «Наука в ЦНТУ: основні досягнення та перспективи розвитку» (2024 р.), основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №15.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Малі затримки й висока швидкість уведення-виводу в IOPS, характерні для твердотільних накопичувачів SSD, роблять їхнім привабливим варіантом для досягнення гарантованої продуктивності. Однак навіть із урахуванням зниження цін на флеш-пам'ять магнітні носії тої ж ємності залишаються набагато більше дешевими, тому з фінансової точки зору повний перехід на флеш-пам'ять у корпоративному середовищі непрактичний. Разом з тим застосування накопичувачів SSD у режимі кешування разом із традиційними обертовими дисками забезпечує високі показники ROI, збалансовану продуктивність і достатню ємність зберігання при прийнятній ціні.

Щоб максимально ефективно використовувати такі якості SSD, як чудовий час доступу й висока швидкість уведення-виводу, важливо вибрати для їхнього розгортання вірне місце в стеці зберігання. Якщо використовувати SSD у верхній частині стека для зберігання «гарячих» даних з високою частотою запитів до них, то в такий спосіб можна домогтися максимальної окупності інвестицій і скоротити число запитів до жорстких дисків.

Так, наприклад, у випадку алгоритму кешування на SSD, використовуваного Qsan, дані в кеші SSD являють собою дублікат часто запитуваної інформації, збереженої на магнітних носіях. При операції читання вони зчитуються з вихідного носія й у фоновому режимі копіюються в кеш SSD, а при наступних запитах читання по тій же адресі можуть бути лічені зі швидких флеш-накопичувачів. Кешуючі алгоритми відрізняються безліччю параметрів, наприклад розміром блоку даних у кеші. Змінюючи ці параметри, кеш SSD можна настроїти для підтримки різних сервісів, таких як файловий сервіс, СУБД або Web-сервіс.

1.2 Область застосування

Використання дедуплікації даних – один з ефективних методів збільшення ROI від інвестицій в SSD, особливо на блоковому рівні, наприклад у системах зберігання SAN або уніфікованих СЗД. (Ще більше збільшити доступну ємність систем зберігання можна за рахунок «тонкого», або динамічного, розподілу ресурсів – Thin Provisioning). Залежно від надмірності дедуплікуємих даних можна легко домогтися трикратної й більше економії ємності зберігання. Зокрема, при застосуванні SSD дедуплікація може виявитися особливо ефективної, якщо відповідну таблицю (Deduplication Data Table, DDT) зберігати у флеш-памяти. При малих затримках доступу до «гарячого» даним, збереженим в DDT, кешування на SSD у сполученні з дедуплікацією може поліпшити використання ємності зберігання й підвищити продуктивність обертових носіїв, і все це – при мінімальних витратах.

Крім застосування в якості кеш-буфера читання, накопичувачі SSD відмінно проявляють себе й при записі. Один такий накопичувач, реалізований у мережній системі зберігання в якості кеша запису, наприклад ZFS Intent Log (ZIL), здатний різко збільшити швидкість синхронних операцій запису в базах даних або в додатках NFS – на 250% при довільній і на 300% при послідовному записі.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Dell Compellent FS8600

Платформа, оптимізована для високої швидкості роботи системи зберігання й низкою сукупної вартості володіння.

FS8600 працює на базі файлової системи нового покоління Dell Fluid File System версії 5 (FluidFS v5). Це одна із самих вигідних доступних масштабованих мережних систем зберігання даних (СЗД) у співвідношенні ціна/продуктивність, що без зайвих витрат забезпечує високе число операцій з файлами в секунду. Це ефективне сполучення дозволяє масштабувати продуктивність і ємність, забезпечуючи наступне:

– Лінійне нарощування продуктивності до 494 000 відкриттів файлів за результатами перевірки в SPEC SFS 2008 і пропускною здатністю до 11,9 Гбіт/с.

– Підвищення продуктивності за допомогою дзеркалювання кеша й автоматизованого балансування навантаження, навіть у міру додавання нових даних.

– До чотирьох мережних систем зберігання даних FS8600 і восьми масивів серії SC в одному масштабованому рішенні для зберігання даних.

Висока щільність у стійці й низькій вартості за гігабайт

Dell Storage Center з масивами FS8600 і SCv2080 забезпечує ідеальну щільність розміщення в стійці для робочих навантажень, пов'язаних з обробкою більших обсягів інформації, і надає наступні переваги:

– **Найвища щільність накопичувачів** у порівнянні з рішеннями п'яти провідних постачальників систем зберігання даних з доступною ємністю в 1 Пбайт у стійці з форм-фактором 14U.

– **Низька вартість за гігабайт** – 0,18 дол. США за гігабайт при доступній ємності для зберігання даних в 1 Пбайт.

– **Підвищена ефективність** завдяки масштабованій архітектурі, оптимізації флеш-накопичувачів, автоматизованому багаторівневому розміщенню й убудованим функціям дедуплікації й стиску на рівні блоків під керуванням політик

– **Відсутність перевантажень** завдяки деталізованому наданню ресурсів і вивільненню місця в мережній системі зберігання даних

Масштабування продуктивності і ємності на льоту

Нарощування продуктивності і ємності системи зберігання файлів в існуючій інфраструктурі, а також прозоре додавання дисків, шасі систем зберігання даних або пристроїв NAS і керуванням ними при необхідності:

– **Зручне масштабування** до 4 Пбайт в одній файловій системі й більше 20 Пбайт в одному просторі імен.

– **Збільшена універсальність** завдяки підтримці декількох протоколів, включаючи SMB, NFS і FTP

– **Локальний захист даних** за допомогою моментальних копій, резервного копіювання NDMP, вилученої реплікації й архівування моментальних копій.

– **Спрощене керування даними** завдяки Dell Storage Manager для керування мережами зберігання даних і мережних систем зберігання даних, квотам файлової системи, фільтрації розширень файлів, а також API керування CLI, Powershell і REST.

– **Підвищення безпеки даних і більше ефективне керування** за рахунок глибокої інтеграції через Active Directory/LDAP, ізолювання мережі й підтримки засобів перевірки файлової системи, наприклад інструмента перевірки змін Change Auditor.

Глобальне обслуговування й підтримка

Послуги Dell допомагають зменшити складність IT-інфраструктури, скоротити витрати й виключити неефективні операції за рахунок оптимізації IT- і бізнес-рішень клієнта. Фахівці Dell по обслуговуванню надають унікальні можливості, включаючи підтримку Copilot для FS8600 з SC4020, SC8000 і SC9000, набір послуг ProSupport для FS8600 для SCv2080, а також рішення по оптимізації й повнім перенесенні даних для прискорення консолідації даних з декількох файлових систем. Фахівці Dell попередньо вивчають всі наявні потреби замовника й проектують рішення відповідно до конкретної інфраструктури й бізнесами-метами. При цьому залучаються місцеві фахівці, ефективно використовуються перевірені методи й велика база знань, що дозволяє скоротити сукупну вартість володіння.

Dell EqualLogic FS7600 і FS7610 з файловою системою FluidFS v4

Нові технології дедуплікації й стиску в системі FluidFS версії 4 дозволяють зменшити ємність, використовувану для зберігання типових даних, до 48 %**.** Технологія скорочення даних Fluid Data також підтримує автоматичний стиск даних, що дозволяє ще більше зменшити ємність для зберігання даних, а, отже, сукупну вартість володіння.

Захист даних за допомогою можливостей корпоративного класу

Файлова система FluidFS v4 також захищає дані завдяки наступним можливостям:

– Моментальні копії на рівні файлів, відновлювані користувачем.

– Асинхронна реплікація на основі моментальних копій на рівні файлової системи на рівноправні системи FS7600 або FS7610.

– Повна реплікація всієї файлової системи, включаючи сховища, експорти й дозволи, щоб спростити відновлення даних після збою

– Антивірусний захист по протоколі CIFS (Common Internet File System) за допомогою антивірусного сервера ICAP і ПЗ сертифікованих партнерів.

– Протокол NDMP (Network Data Management Protocol) для резервного копіювання й відновлення після збоїв.

Інтелектуальне керування ростом

Керування ростом ємності системи зберігання даних завдяки гнучким можливостям масштабування ємності й продуктивності без переривання роботи систем:

– Простота об'єднання з існуючими або новими масивами EqualLogic серії PS, а також виключення глобальних відновлень.

– **Збільшення ємності блокової й файлової системи зберігання даних** усередині єдиного простору імен до 509 Тбайт завдяки серверному компоненту системи зберігання даних масивів EqualLogic PS.

– **Підвищення масштабованості, продуктивності й безпеці** завдяки розширеній підтримці протоколів.

– **Просте підвищення продуктивності** завдяки установці другого пристрою FS7600 або FS7610.

– **Інтуїтивно зрозуміле керування операціями з файлами й блоками** за допомогою EqualLogic Group Manager або інтерфейсу командного рядка.

Створення інтегрованої й надійної мережі

Створіть повністю інтегровану комплексну універсальну систему зберігання на базі системи EqualLogic завдяки пристроям мережних систем зберігання даних FS7600 і FS7610. Ці рішення працюють із масивами EqualLogic серії PS і дозволяють реалізувати основні переваги платформи EqualLogic, включаючи однорангову архітектуру, прості у використанні функції, а також ліцензії на повнофункціональне програмне забезпечення. Система EqualLogic серії FS надає наступні можливості:

– **Спеціальна платформа мережної системи зберігання даних з форм-фактором 2U** із двома високодоступними контролерами в конфігурації "активний-активний" і магістраллю PCIe для обміну даними між контролерами.

– **FS7610: чотири порти 10Gb** для підключення до мережі зберігання даних і чотири порти для клієнтської мережі.

– **FS7600: вісім портів Gb для підключення до мережі зберігання даних** і вісім портів для клієнтської мережі.

Для забезпечення надійності щодня пристрою NAS EqualLogic FS7600 і FS7610 надають наступні функції:

– **Кластеризація в режимі " активний-активний"** і дзеркалювання кеша для пар контролерів.

– **Убудоване резервне джерело живлення** для захисту кешованих даних у випадку збоїти контролера

– **Контролери, джерела живлення й вентилятори** з повним резервуванням і можливістю гарячого перемикання**.**

– **Автоматичне перемикання контролера при відмові**.

– **Контрольні суми й відказостійка реєстрація** для захисту метаданих файлів.

Глобальне обслуговування й підтримка

Зменшення складності IT-інфраструктури, скорочення витрат і виключення неефективних операцій за рахунок оптимізації IT- і бізнес-рішень клієнта. Корпорація Dell надасть вам комплексне рішення, що максимально підвищить продуктивність і час безперебійної роботи ваших систем. Рішення й послуги Dell для підприємств, що лідирують в області серверів, систем зберігання даних і мереж, надають інновації в будь-якому масштабі. Якщо ж ви хочете заощадити або підвищити ефективність роботи, фінансові послуги Dell надають різноманітні асортименти варіантів для простого й недорогого придбання технологій.

Послуги Dell ProSupport

Dell ProSupport Plus для критично важливих систем або Dell ProSupport для підтримки встаткування й програмного забезпечення преміум-класу рекомендуються для забезпечення оптимальної підтримки рішення EqualLogic. Для одержання додаткової інформації звернетеся до торговельного представника Dell.

Комплексне навчання. Комплексні навчальні курси дозволяють одержати навички, необхідні для того, щоб почувати себе впевнено в умовах високої конкуренції на ринку IT. Dell пропонує найрізноманітніші курси під керівництвом інструктора, а також інтерактивні курси навчання по різних темах – від обслуговування серверів до докладного вивчення технічних аспектів конкретних систем.

Сімейство мережних пристроїв зберігання даних **(NAS) Dell Storage NX**

Удосконалена мережна система зберігання даних.

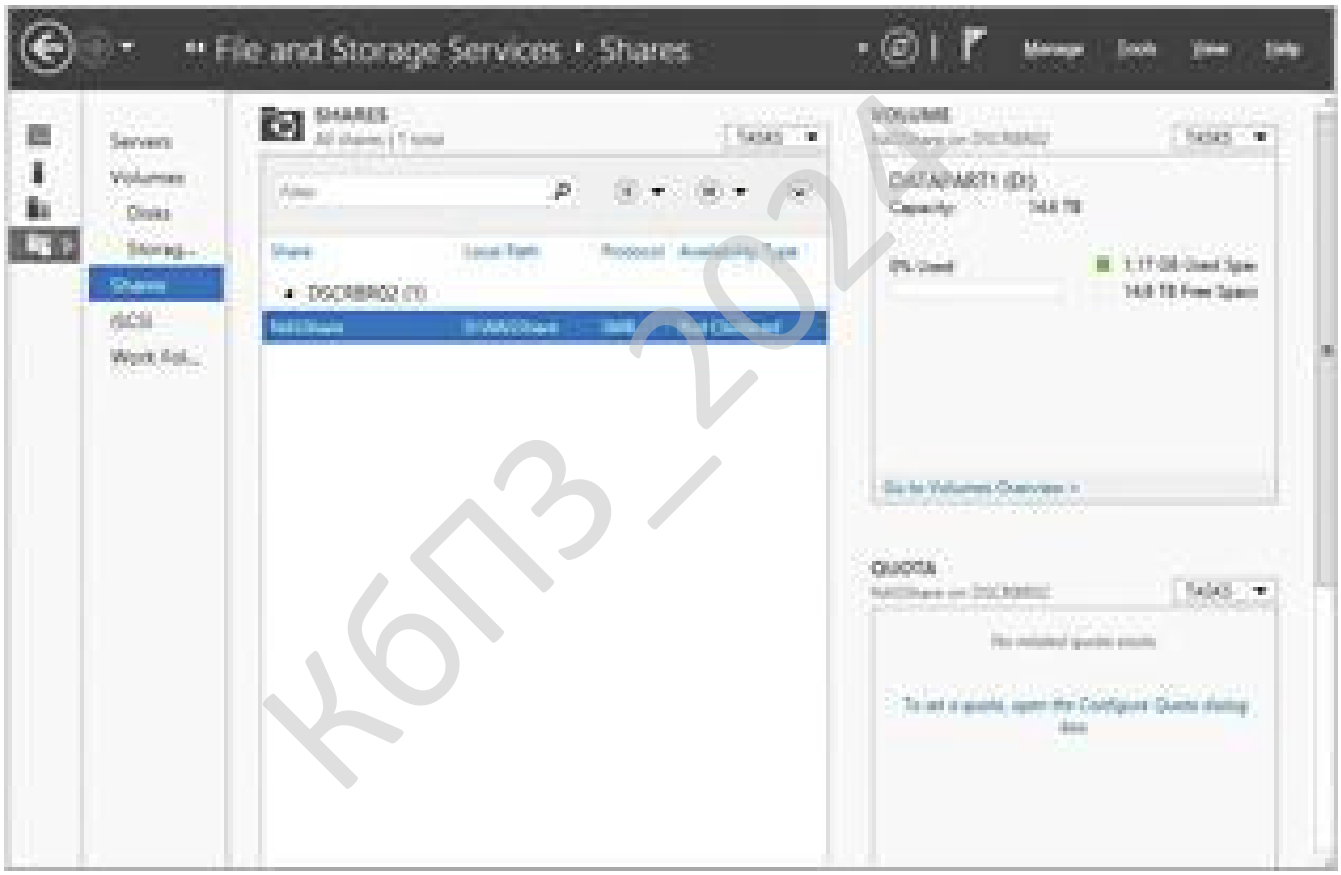

Рисунок 2.1 – Інтерфейс користувача мережних пристроїв зберігання даних (NAS) Dell Storage NX

Пристрої зберігання даних Dell Storage NX прості у використанні й керуванні завдяки добре знайомому централізованому інтерфейсу керування для автономних і кластерних конфігурацій:

– **Попередньо встановлена ОС** для простоти розгортання.

– **Оптимізація для роботи з файлами в**Windows Storage Server 2012 R2.

– **Функції кластеризації** при використанні NX3330.

Платформи PowerEdge

Покладетеся на високу швидкість і ефективність при використанні пристроїв NAS на базі серверної технології PowerEdge 13-го покоління (сервери встановлюються в стійку).

– Багатоядерні процесори Intel® Xeon® нового покоління.

– До 32 Гбайт пам'яті й номінальна ємність до 72 Тбайт (NX3230).

– Варіанти з жорсткими дисками SATA, SAS і NL-SAS.

Можливість використання шлюзу для мережної системи зберігання даних

Установите NX3330 як шлюз мережної системи зберігання даних і користуйтеся додатковими масивами мереж зберігання даних для широкомасштабного зовнішнього розширення. Підтримка кластеризації (до 64 вузлів) з використанням томів Clustered Shared Volumes (CSV) при підключенні до масивів зберігання PowerVault MD3, EqualLogic або Dell Compellent.

Просте керування

Затрачайте менше часу на керування даними й більше часу на розвиток бізнесу завдяки Microsoft® System Center і вилученому робітникові столу для функції адміністрування в Windows Storage Server 2012 R2 або керуванню даними за допомогою убудованого контролера вилученого доступу Dell iDRAC7 Enterprise Edition. Крім того, ви можете підвищити ефективність використання ємності мережної системи зберігання даних і оптимізувати дисковий простір за допомогою наступних убудованих функцій пристроїв сімейства Dell Storage NX:

– Дедуплікація даних.

– Деталізоване надання ресурсів.

– Можливості синхронізації DFS-R у мережних підключеннях з обмеженою пропускною здатністю в розподіленому й вилученому офісному середовищах.

– Служба тіньового копіювання томів (VSS) для загальних файлових папок SMB, що дозволяє створювати резервні копії операцій з використанням моментальних знімків вилучених загальних файлових папок, що підтримують серверні додатки на основі SMB.

Оновлені ОС

Сімейство Dell Storage NX працює на базі програмного забезпечення Windows® Storage Server 2012 R2, що забезпечує продуктивність і багатофункціональність Microsoft Windows Server® 2012 R2 у роботі пристроїв NAS. Це наступні продукти:

– Новітні функції SMB 3.0, що підтримують Hyper-V і сервер SQL.

– Кластеризація, що забезпечує безперервну доступність і відказостійкість, з використанням NFS і iSCSI.

– Автоматичне виявлення й використання декількох мережних підключень між клієнтом і сервером SMB.

Глобальне обслуговування й підтримка

Послуги Dell допомагають зменшити складність IT-інфраструктури, скоротити витрати й виключити неефективні операції за рахунок оптимізації IT- і бізнес-рішень клієнта. Група фахівців з обслуговування Dell пропонує унікальні можливості, включаючи рішення для повного перенесення даних, які дозволяють спростити консолідацію декількох файлових систем. Фахівці Dell попередньо вивчають всі потреби клієнта й проектують рішення відповідно до конкретного середовища й бізнес-цілями. При цьому залучаються місцеві фахівці, ефективно використовуються перевірені методи й великі знання в конкретній області, що дозволяє скоротити сукупну вартість володіння.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програмне забезпечення написане мовою Visual C#. Ця мова обрана виходячи з наступних міркувань. Visual C# – строго типізована об'єктно-орієнтована мова, призначена для розробки різноманітних безпечних і потужних додатків, виконуваних у середовищі .NET Framework. Мовою Visual C# можна розробляти звичайні клієнтські додатки Windows, веб-служби XML, розподілені компоненти, додатки типу " сервер-клієнт", додатки баз даних і багато яких інших. В Visual C# 2012 є розширений редактор коду, конструктори зі зручним користувальницьким інтерфейсом, вбудований відладник і багато інших засобів, покликані спростити розробку додатків мовою Visual C# версії 5.0 і .NET Framework версії 4.5.

Синтаксис Visual C# дуже виразний, але простий у вивченні. Усі, хто знаком з мовами C, C++ або Java з легкістю взнають синтаксис із фігурними дужками, характерний для мови Visual C#. Розроблювачі, що знають кожну із цих мов, як правило, зможуть домогтися ефективної роботи з мовою Visual C# за дуже короткий час. Синтаксис Visual C# робить простіше те, що було складно в C++, і забезпечує потужні можливості, такі як типи значень Nullable, перерахування, делегати, лямбда-вираження й прямий доступ до пам'яті, чого немає в Java. Visual C# підтримує універсальні методи й типи, забезпечуючи більше високий рівень безпеки й продуктивності, а також ітератори, що дозволяють при реалізації колекцій класів визначати власне поводження ітерації, що може легко використовуватися в клієнтському коді. В Visual C# 5.0 вираження LINQ (Language-Integrated Query) роблять строго-типізований запит першокласною конструкцією мови.

Як об'єктно-орієнтована мова, Visual C# підтримує поняття інкапсуляції, спадкування й поліморфізму. Всі змінні й методи, включаючи метод `Main` – точку входу додатка – інкапсулюється у визначення класів. Клас може

успадковувати безпосередньо з одного родового класу, але може реалізовувати будь-яке число інтерфейсів. Для методів, які перевизначають віртуальні методи в батьківському класі, необхідно ключове слово override, щоб виключити випадкове повторне визначення. У мові Visual C# структура схожа на полегшений клас: це тип, що розподіляється по стопках, що реалізує інтерфейси, але не підтримуюче спадкування.

На додаток до основних описаних об’єктно-орієнтованих принципів, мова Visual C# спрощує розробку компонентів програмного забезпечення завдяки декільком інноваційним конструкціям мови, у число яких входять наступні:

– Інкапсульовані підписи методів, називані делегатами, які підтримують строго-типізовані повідомлення про події.

– Властивості, що виступають у ролі методів доступу для закритих змінних-членів.

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

– LINQ (Language-Integrated Query), що пропонує вбудовані можливості запитів у різних джерелах даних.

Якщо буде потрібно забезпечити взаємодію з іншим програмним забезпеченням Windows, таким як об'єкти COM або власні бібліотеки DLL Win32, у мові Visual C# можна використовувати процес, що називається "Interop". Процес Interop дозволяє програмам на Visual C# виконувати практично будь-які дії, які може виконувати вихідний додаток на C++. Мова Visual C# підтримує навіть покажчики й поняття "небезпечного" коду для тих випадків, коли прямий доступ до пам'яті має вкрай важливе значення.

Процес побудови Visual C# у порівнянні з C і C++ простий і є більше гнучким, чим в Java. Немає окремих файлів заголовка, а методи й типи не потрібно повідомляти в певному порядку. У вихідному файлі Visual C# може бути визначене будь-яке число класів, структур, інтерфейсів і подій.

Архітектура платформи .NET Framework

Програма мовою Visual C# виконується в середовищі .NET Framework – інтегрованому компоненті Windows, що містить віртуальну систему виконання (середовище CLR) і уніфікований набір бібліотек класів. Середовище CLR являє собою комерційну реалізацію корпорацією Майкрософт інфраструктури CLI, що є міжнародним стандартом, який лежить в основі створення середовищ виконання й розробки, у яких забезпечується тісна взаємодія між мовами й бібліотеками.

Вихідний код, написаний мовою Visual C#, компілюється в проміжну мову (IL) у відповідності зі специфікацією CLI. Код IL і ресурси, такі як растрові зображення й рядки, зберігаються на диску у файлі, що виконується, називаному складанням, з розширенням EXE або DLL у більшості випадків. Складання містить маніфест із відомостями про типи складання, версії, мови й регіональні параметри та вимоги безпеки.

При виконанні програми на Visual C# складання завантажується в середовище CLR залежно від відомостей у маніфесті. Далі, якщо вимоги безпеки дотримані, середовище CLR виконує JIT-компіляцію для перетворення коду IL в інструкції машинного коду. Середовище CLR також надає інші служби, що відносяться до автоматичного збору сміття, обробки виключень і керуванню ресурсами. Код, виконуваний середовищем CLR, іноді називають "керованим кодом" у протиставлення "некерованому коду", що компілюється в машинний код, призначений для певної системи. Далі показані відносини під час компіляції й час виконання між файлами з вихідним кодом Visual C#, бібліотеками класів .NET Framework, складаннями й середовищем CLR.

Взаємодія між мовами є ключовою особливістю .NET Framework. Оскільки код IL, створюваний компілятором Visual C# відповідає специфікації CTS, код IL на основі Visual C# може взаємодіяти з кодом, створюваним версіями мов Visual Basic, Visual C++, Visual J# платформи .NET Framework і ще більш ніж 20 CTS-сумісних мов. В одному складанні може бути кілька модулів, написаних

на різних мовах платформи .NET Framework, і типи можуть посилатися один на одного, як якби вони були написані на одній мові.

Крім служб часу виконання, в.NET Framework також є велика бібліотека, що складається з більш ніж 4000 класів, організованих по просторах імен, які забезпечують різноманітні корисні функції для будь-яких дій, починаючи від введення й виведення файлів для керуванням рядками для розбивки XML, і закінчуючи елементами керування Windows Forms. У звичайному додатку мовою Visual C# бібліотека класів .NET Framework інтенсивно використовується для "устрою" коду.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) 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 контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Включення у свій час програмного забезпечення ініціатора iSCSI в Windows 7 і Windows Server 2008 вплинуло на корпоративні системи зберігання, дозволивши без додаткових зусиль реалізувати блоковий доступ до мережних СЗД не тільки на серверах, але й на звичайних настільних ПК і робочих станціях. Однак простота використання iSCSI виявилася ціпком про два кінці. Хоча сумісність програмного забезпечення iSCSI – зручна властивість сучасних операційних систем, програмна обробка команд зберігання даних iSCSI і пакетів TCP/IP вимагає ресурсів процесора хосту, що знижує його продуктивність.

Рішення складається у використанні апаратних засобів для «розвантаження» iSCSI і TCP/IP (offloading). При виділенні процесорного ядра для обробки даних iSCSI і TCP/IP, апаратні засоби, такі як Qsan QiSOE, можуть при мінімальних початкових інвестиціях знизити непродуктивні витрати на протокол, що виливається в значне поліпшення продуктивності як при послідовних читанні й записі, так і при довільному доступі до даних.

Накладні витрати на комунікаційний протокол позначаються як на хості, так і на пристрої зберігання. Апаратні карти розширення на стороні хосту – так звані конвергентні мережні адаптери (Converged Network Adapter, CNA) – поєднують функціональність традиційної мережної карти й адаптера шини хосту (Host Bus Adapter, HBA), переносячи обробку протоколів TCP/IP і iSCSI в устаткування. Виключення обробки протоколу зберігання даних із програмного навантаження виявляється корисним і в інших випадках. Протокол Fibre Channel, найбільш популярний у мережах зберігання SAN, теж може одержати значний виграш по продуктивності за рахунок переносу на апаратний рівень непродуктивних витрат при обробці протоколу зберігання даних.

Апаратний RAID

Забезпечуючи надмірність даних, контроль парності й навіть чудову продуктивність, RAID є однієї з основних технологій систем зберігання даних корпоративного рівня, про яку проте нерідко забувають і не враховують того, що програмний RAID позначається на продуктивності. Будучи убудованим у кожну сучасну операційну систему, він не завжди ідеально відповідає потребам корпоративних ЦОД.

Відділення реалізації RAID від операційної системи може дати ряд переваг. При простої реалізації, такий як RAID0 або RAID1, впливом програмного забезпечення RAID на продуктивність системи можна зневажити, але цей режим рідко застосовується в корпоративних системах. Якщо враховувати операції перевірки парності й велика кількість дисків, то апаратний RAID буде помітно відрізнятися від програмного в характеристиках IOPS.

Апаратний RAID більше надійний для корпоративного застосування. На випадок проблем з електроживленням передбачена можливість використання батареї (Battery Backup Unit, BBU), що дозволяє зберігати всі важливі дані в енергонезалежній пам'яті й зчитувати їх звідти після подачі електроенергії. Оскільки керування RAID відділене від операційної системи, цілісності даних ніщо не загрожує навіть при аварійному завершенні роботи ОС. Крім того, апаратний RAID здатний захищати дані в процесі завантаження операційної системи, у той час як програмний «виходить на сцену» лише по його завершенні.

Хочемо ми того чи ні, виходять із ладу навіть диски корпоративного класу. У такому випадку RAID повинен перешикувати масив, використовуючи диск гарячого резерву й дані парності зі справних дисків. Якщо ємність виміряється в гігабайтах, час перебудування дискового масиву виявиться цілком прийнятним, але при терабайтної ємності, що зараз уже не рідкість, інтервал перебудування може становити кілька годин, і все це час корпоративна інформація буде уразливою. У сучасних ЦОД, де ємності зберігання, звичайно, виміряються терабайтами й петабайтами, функція швидкого перебудування (fast rebuild), убудована в мережні СЗД, є вкрай необхідною.

У корпоративних ЦОД широко застосовується віртуалізація. У мережних СЗД віртуалізуються ресурси зберігання, що дозволяє управляти ними централізовано. Крім того, завдяки віртуалізації можна консолідувати обчислювальні потужності, централізовано управляти робочими станціями, відновленнями й т.д. Всі ці переваги цілком реальні, але найбільші вигоди приносить використання мережних систем зберігання даних, сертифікованих для віртуалізованих середовищ.

Тим часом віртуалізація пред'являє до систем зберігання унікальні вимоги, і СЗД нерідко стають вузьким місцем при спробі збільшити продуктивність середовища VDI. Так, VMware, беручи до уваги унікальні вимоги, які віртуалізація пред'являє до СЗД, пропонує спеціальні протоколи, наприклад VMware vSphere Storage API – Array Integration (VAAI). Аналогічно тому, як операції iSCSI і RAID можна перенести на рівень спеціальних апаратних засобів, тим самим розвантаживши процесор, VAAI дозволяє відокремити операції зберігання даних від сервера віртуалізації. У результаті кожний компонент ЦОД одержує можливість робити те, на що він найбільше здатний. Устаткування зберігання даних, сертифіковане й сумісне з VAAI, не тільки підвищує продуктивність середовища віртуалізації, але й забезпечує ефективне виконання найважливіших функцій віртуалізованих СЗД – зокрема, клонування, обнуління й знімки даних.

Ключовими метриками будь-якого корпоративного ЦОД є продуктивність і надійність мережних СЗД, але диявол криється в деталях. Кешування на SSD у багаторівневому середовищі зберігання (tiered storage) і динамічний розподіл ємності (thin provisioning), перенос функцій iSCSI і RAID на апаратний рівень (offloading), вибір систем зберігання даних з функцією швидкого перебудування (fast rebuild), сумісність із VMware дозволять компаніям одержати максимальну віддачу від своїх інвестицій в устаткування.

3.2 Розробка структурної схеми

Мережна система зберігання даних (NAS) – це пристрій зберігання із загальним доступом, що надає консолідовану файлову систему й послуги зі зберігання серверам відкритих систем. Додатки й користувачі одержують доступ до даних через загальну IP-мережу. У кожного NAS-пристрою є власна унікальна IP-адреса.

Як працює NAS

NAS – це відкрита комп'ютерна система з ємністю для зберігання, приєднаної до мережі, що надає послуги зі зберігання даних на файловій основі іншим пристроям у мережі. NAS використовує стандартну файлову систему CIFS і мережну файлову систему NFS, що дозволить клієнтам Microsoft Windows, Linux і UNIX одержати доступ до файлів, файловим системам і базам даних в IP-мережі.

Переваги NAS

Коли в наявності є більші обсяги, різні типи й рівні системи зберігання даних, приєднаної безпосередньо до сервера (DAS) або внутрішньої СЗД, може бути складно й дорого управляти даними й підтримувати їх у зв'язку з дуже низькими коефіцієнтом використання. NAS надає економію засобів і простоту в консолідованому зберіганні при використанні існуючої IP-мережі, при цьому усуває витрати й складності, зв'язані з будівництвом і підтримкою вторинної мережі, як з мережею зберігання даних (SAN).

Можна виділити три способи рішення завдання побудови каскаду схоронності даних.

1. Мережне резервування на інші сервера й NAS. Мережне резервування має масу переваг, тому що дозволяє резервувати масив даних незалежно від дистанції до мети зберігання резервної копії. Мережний сервер зберігання резервних копій може одночасно використовуватися й для резервування інших типів дані компанії. При мережному резервуванні процес передачі потоку даних

може відбуватися по протоколах: FTP, sFTP, HTTP/HTTPs, а також rsync, що найбільше часто зустрічається для міжсерверного обміну.

Якщо як базове встаткування відеореєстрації й зберігання використовуються NAS-сервери, то між джерелом і метою резервування можна створити не просто систему автоматичного резервування за розкладом, але й автосинхронізацію. Тобто в режимі реального часу безперервним потоком буде відбуватися дзеркалювання даних, створюючи дублюючий образ масиву на іншому NAS, установленому в межах вашої локальної мережі або десь у вилученому офісі На сьогодні мережне резервування є самим вірним рішенням для організації постійного процесу резервування даних. Мінусом такого рішення можна назвати вартість, оскільки вартість резервного сервера або NAS може наближатися до вартості провідного сервера. Серед провідних виробників спеціалізованих мережних систем зберігання й резервування можна відзначити наступних найбільш помітних гравців:

– середній рівень і початок верхнього – Netgear, Synology, QNAP;

– масштабовані системи вищої категорії (причому й у плані ціни) – Netapp, EMC2, HP.

2. Локальне резервування на NAS. Одержавши у своє розпорядження 8 – 10 дискових накопичувачів, ви неминуче зштовхнетеся з такою проблемою: куди можна зарезервувати настільки великий обсяг даних. Якщо з якихось причин немає можливості використовувати аналогічне по ємності мережне сховище, то відповіддю є NAS (Direct Attached Storage) – зовнішні сховища прямого підключення.

NAS являє собою апаратний RAID-масив без якої-небудь убудованої ОС. Управляється й конфігурується зовнішніми засобами Це автономний продукт ємністю звичайно від 4 до 8 дисків і більше, створений для безпосереднього підключення до сервера, NAS або NVR як зовнішня ємність розширення або зовнішнього масиву, на який здійснюється локальне резервування даних із провідного сервера або NAS.

Перевагою NAS є ціна, що звичайно в 1,5 рази нижче NAS з ідентичною кількістю дисків. NAS можуть підключатися до сервера або NVR по інтерфейсах eSATA, USB 2.0/3.0 або SAS і підтримують рівні зберігання JBOD, RAID5, 50, 0 і касетний режим. Якщо ми підключимо NAS до будь-якому NAS (навіть початкового рівня) або серверу й пропишемо відповідні права доступу до цієї зовнішньої ємності, то одержимо для цілей резервування справжнє мережне сховище високої ємності, причому за дуже розумні гроші.

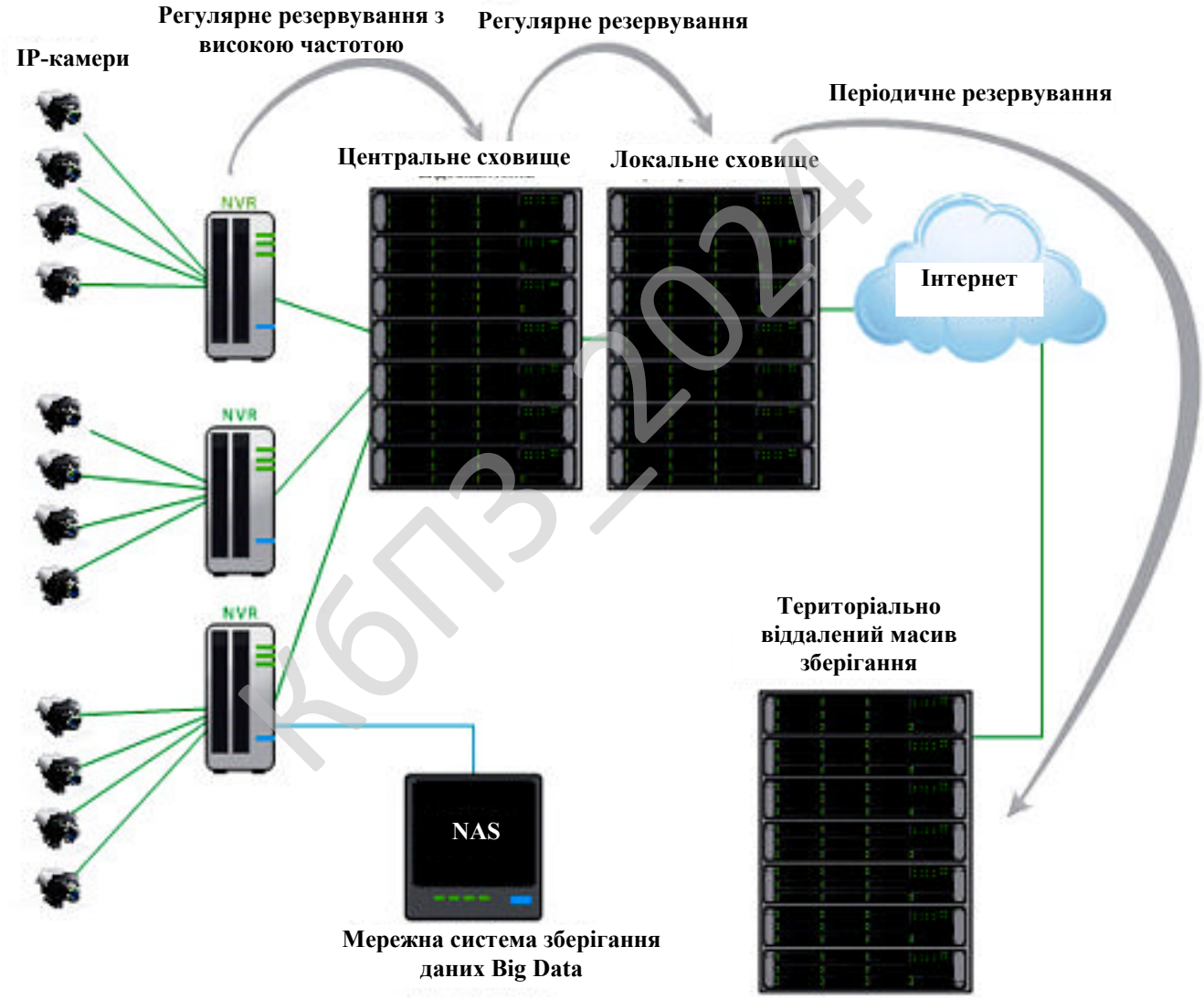

Рисунок 3.1 – Структурна схема системи

Серед виробників NAS-накопичувачів можна відзначити CFI Group, RAIDON, Promise Technology. У виробника мережних систем зберігання Synology є фірмове рішення – модулі розширення DX/RX. По суті, це теж NAS-продукти, але розраховані винятково для підключення до мережних накопичувачів і NVR Synology Особливість даного рішення – можливість об'єднання даних у єдиний масив зберігання з головним NAS.

3. Резервування в комерційні ЦОД. Як варіант реалізації територіально розподіленого резервування в додатку до завдання схоронності особливо коштовного відеоконтенту можна розглядати систему резервування в комерційні ЦОД. Безумовно, сьогодні це недешево, і як і раніше можуть виникати проблеми з каналами передачі даних, але, як правило, завдання організації постійного каналу між вашим власним центром агрегації відеоконтенту й зовнішнім ЦОД вирішити можливо. Особливо якщо жорстко ставити завдання виключення більшості ризиків, пов'язаних з локалізацією даних у межах власних мереж зберігання.

Для зв'язку із ЦОД може бути використана й фізична лінія, і передача даних через Інтернет за умови організації VPN-підключення до ЦОД. VPN забезпечує захищеність каналу й дає можливість мережним накопичувачам і серверам звертатися до ємностей ЦОД як до локальних ресурсів своєї мережі. Це забезпечує можливості застосування стандартних політик резервування. Сучасні NVR, наприклад Synology, мають убудований VPN-клієнт і можуть самостійно створювати тунель резервування в дата-центр.

Яку би модель по забезпеченню схоронності даних не вибрала ваша компанія, вона повинна бути попередньо ретельно осмислена й розрахована математично як з погляду необхідних ємностей, так і з погляду циклу відновлення масиву. Сучасні технології інкрементального резервування й зберігання снапшотів (не самого репліканта масиву, а його моментальних знімків на точку часу) можуть істотно знизити потреба в обсязі зберігання й, отже, зберегти ваші гроші.

Резервування – дуже важливий і непростий процес як з погляду правильної технічної реалізації, так і з погляду організаційній складовій. Рекомендується обов'язково мати чітко формалізовану схему політики резервування й розписати по ролях всі зони персональної відповідальності фахівців з екстреного відновлення даних з резервних ємностей і введенню системи в роботу у випадку збоїти

Аудит мережної інфраструктури

Будь-які мережні пристрої відеореєстрації будуть працювати в межах конкретної локальної мережі з усіма її проблемами й недоліками. Задайте собі питання, чи може поточна мережна інфраструктура стати причиною збоїти й розриву зв'язку між камерами й NVR? На жаль, часто відповідь може бути позитивним. Не заглиблюючись у тему причин і недоліків, опишемо, що потрібно мати на увазі для зниження ризиків.

1. Мережа відеоспостереження повинна бути відділена від мережі загального користування. При використанні безлічі камер високого дозволу пропускна здатність мережі може досягати граничних значень, тому не виключається негативний взаємний вплив офісного трафіку й відеопотоків з камер, особливо у великих організаціях. Крім того, якщо з вини IT-адміністратора трапляється тимчасовий колапс у мережі, можуть бути зроблені перезавантаження мережного встаткування й зв'язаних серверів без попередження адміністрації мережі відеоспостереження. В ідеалі в IT-устаткування й устаткування мережі відеоспостереження повинні бути різні відповідальні особи з поділом прав доступу й навіть різна локалізація

2. При підключенні до активного мережного встаткування NVR урахуйте їхню особливість: сучасні NVR мають кілька мережних портів (звичайно 2 – 4). Включайте NVR різними портами в різні комутатори (світчі), тоді навіть вихід з ладу одного з комутаторів не паралізує відеореєстрацію повністю Ніколи не включайте всі камери одного сегмента відеоспостереження в один свитч або хаб.

3. Зареєструйте камери, установлені в критично важливих зонах відеоспостереження, на декількох NVR, що мають підключення до різного мережного встаткування. Більшість професійних IP-камер мають функцію генерації декількох потоків одночасно. Збережете таблицю крос-реєстрації камер на різних NVR.

Аудит системи енергопостачання й резервного живлення

Велика історія СЗД і систем відеореєстрації знала масу прикладів, коли, вклавши масу грошей в устаткування й ПЗ, люди відкладали "на потім" яку-небудь на перший погляд несуттєву деталь – систему кондиціювання або резервного живлення. І те й інше найчастіше приводило до застрашливої тиші в серверному й організаційно-кадровому виводам керівництва. І якщо кондиціювання – річ досить дорога, її хочеться мати, але аргументувати її закупівлю керівництву не завжди вдається, то за безперебійне живлення у випадку СЗД і систем збору відеоконтенту потрібно просто битися

Необхідно взяти за правило, що в системах СЗД і відеореєстрації кожний вузол повинен мати хоча б примітивне джерело резервного живлення – хоча б побутовий ДБЖ. Джерела безперебійного живлення (ДБЖ) повинні бути забезпечені кожному NVR, на який відбувається збір хоч скільки-небудь коштовної інформації. ДБЖ повинні живити світчі й варті перед ними маршрутизатори. Втрата живлення на камері може відключити відеоканал на якийсь час, а різкий кидок напруги або провал по живленню можуть віднести із собою в "кращий світ" десятки терабайт важливого відеоархіву разом з NVR, який вийшов з ладу.

Необхідно згадати ще про одну особливість: багато джерел безперебійного живлення прекрасно працюють при тимчасових відключеннях живлення, але найчастіше самі виявляються жертвами у випадку різкого стрибка напруги, тому що тільки деякі моделі ДБЖ мають убудовані стабілізатори напруги. Тому, якщо ви знаєте, що у вас може "гуляти" напруга, необхідно затурбуватися захистом ланцюга живлення потужним стабілізатором напруги.

Треба пам'ятати, що все, що пов'язане із системами безпеки, повинне мати багаторазове дублювання.

Треба визнати, що вибір джерел резервного живлення невеликий як для традиційних серверів, так і для NVR. На мій погляд, найбільш вірним вибором буде ДБЖ від APC, тому що вони мають підтримку зворотного зв'язка з NVR. Як пристрої для захисту мережного встаткування підійде й APC, і Powercom, і в принципі будь-який іншої. Для енергозахисту великих систем підійдуть продукти Emerson, але отут, звичайно, питання бюджету.

Те ж саме й для Windows-серверів: найбільш удалі програмні пакети по керуванню живленням сервера й екстреним відключенням і перезапуском при поновленні живлення створені саме для продуктів APC.

У ряду сучасних серверів доступне налаштування широкомовного оповіщення "побратимів" про надходження сигналу тривоги з ДБЖ. У результаті може бути активований ланцюговий механізм безпечного гасіння NAS-серверів.

Незважаючи на наявність всіх функцій керування живленням і в NVR/NAS, і в традиційних серверах, не варто зневажати такою простою функцією, як налаштування політик засипання й відключення у випадку одержання сигналу тривоги. Якщо ви не визначите їх, то встаткування саме "не догадається".

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно. З неї бачимо, що розроблене програмне забезпечення системи підвищення надійності зберігання даних на сервері складається з наступних основних функціональних блоків:

– сам носій інформації – сервери мережних систем зберігання;

– кодер Ріда-Соломона, який використовується, коли відбувається запис інформації на сервер, при цьому необхідно враховувати, що об’єм інформації, яка

записується на диск серверу, повинна бути меншою, ніж об’єм диску, в зв’язку, з тим, що коди Ріда-Соломона відносяться до кодів з надмірністю, за рахунок якої й відбувається кодування;

– декодер Ріда-Соломона, який використовується при читанні даних с відповідного диску.

Функціонально блок, який утримує кодер Ріда-Соломона включає в себе наступні блоки:

– дані, які потрібно зберегти на сервері мережних систем зберігання;

– поліном для кодування;

– відмовостійки коди.

Коди Ріда-Соломона базуються на спеціальному розділі математики – полях Галуа (GF) або кінцевих полях. Арифметичні дії (+, -, х, / і т.д.) над елементами кінцевого поля дають результат, що також є елементом цього поля. Кодер та декодер Ріда-Соломона повинні вміти виконувати ці арифметичні операції. Ці операції для своєї реалізації вимагають спеціального устаткування або спеціалізованого програмного забезпечення.

Кодове слово Ріда-Соломона формується із залученням спеціального полінома. Всі коректні кодові слова повинні ділитися без залишку на ці утворюючі поліноми. Загальна форма утворюючого полінома має вигляд: $g(x) = (x-a^{i})(x-a^{i+1})\ldots(x-a^{i+2t})$, а кодове слово формується за допомогою операції: $c(x) = g(x).i(x)$, де $g(x)$ є утворюючим поліномом, $i(x)$ являє собою інформаційний блок, $c(x)$ – кодове слово, що називається простим елементом поля.

2t символів парності в кодовому слові Ріда-Соломона визначаються з наступного співвідношення: $p(x) = i(x)\cdot x^{n-k} \bmod g(x)$.

Застосування

У даний момент коди Ріда-Соломона мають дуже широку область застосування завдяки їхній здатності знаходити й виправляти багаторазові пакети помилок.

Код Ріда-Соломона використовується при записі й читанні в контролерах

оперативної пам'яті, при архівуванні даних, запису інформації на жорсткі диски (ECC), запису на CD/DVD диски.

Навіть якщо ушкоджено значний обсяг інформації, зіпсовано кілька секторів дискового носія, то коди Ріда-Соломона дозволяють відновити більшу частину загубленої інформації. Також використовується при записі на такі носії, як магнітні стрічки й штрихкоди.

Можливі помилки при читанні з диска з'являються вже на етапі виробництва диска, тому що зробити ідеальний диск при сучасних технологіях неможливо.

Так само помилки можуть бути викликані подряпинами на поверхні диска, пилом і т.д. Тому при виготовленні компакт-диску, що читається, використовується система корекції CIRC (Cross Interleaved Reed Solomon Code). Ця корекція реалізована у всіх пристроях, що дозволяють зчитувати дані з CD дисків, у вигляді чипа із прошиванням firmware. Знаходження й корекція помилок заснована надмірності й перемежені (redundancy & interleaving).

Надмірність приблизно 25% від вихідної інформації.

При записі на цифрові аудіокомпакт-диски (Compact Disc Digital Audio – CD-DA) використовується стандарт Red Book.

Корекція помилок відбувається на двох рівнях C1 і C2.

При кодуванні на першому етапі відбувається додавання перевірочних символів до вихідних даних, на другому етапі інформація знову кодується.

Крім кодування здійснюється також перемішування (перемеження) байтів, щоб при корекції блоки помилок розпалися на окремі біти, які легше виправляються.

На першому рівні виявляються й виправляються помилкові блоки довжиною один і два байти (один і два помилкових символи відповідно). Помилкові блоки довжиною три байти виявляються й передаються на наступний рівень.

На другому рівні виявляються й виправляються помилкові блоки, що виникли в C2, довжиною 1 і 2 байти. Виявлення трьох помилкових символу є

фатальною помилкою, не можуть бути виправлені.

Цей алгоритм кодування використовується при передачі даних по мережах WiMAX, в оптичних лініях зв'язку, у супутниковому й радіорелейному зв'язку. Метод прямої корекції помилок у минаючому трафіку (Forward Error Correction, FEC) ґрунтується на кодах Ріда-Соломона.

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

Перейдемо до розгляду іншого функціонального блоку – декодеру Ріда-Соломона.

Цей функціональний блок містить у собі наступні блоки:

– відмовостійкий код;

– блок обчислення поліномів синдрому та помилок;

– блок обчислення локації та значень помилок;

– блок корекції помилок;

– відновлені за допомогою коді Ріда-Соломона дані.

Декодер працює наступним чином.

Введемо позначення:

– r(x) – Отримане кодове слово.

– Si – Синдроми.

– L(x) – Поліном локації помилок.

– Xi – Положення помилок.

– Yi – Значення помилок.

– c(x) – Відновлене кодове слово.

– v – Число помилок.

Отримане кодове слово r(x) являє собою вихідне (передане) кодове слово c(x) плюс помилки: r(x) = c(x) + e(x).

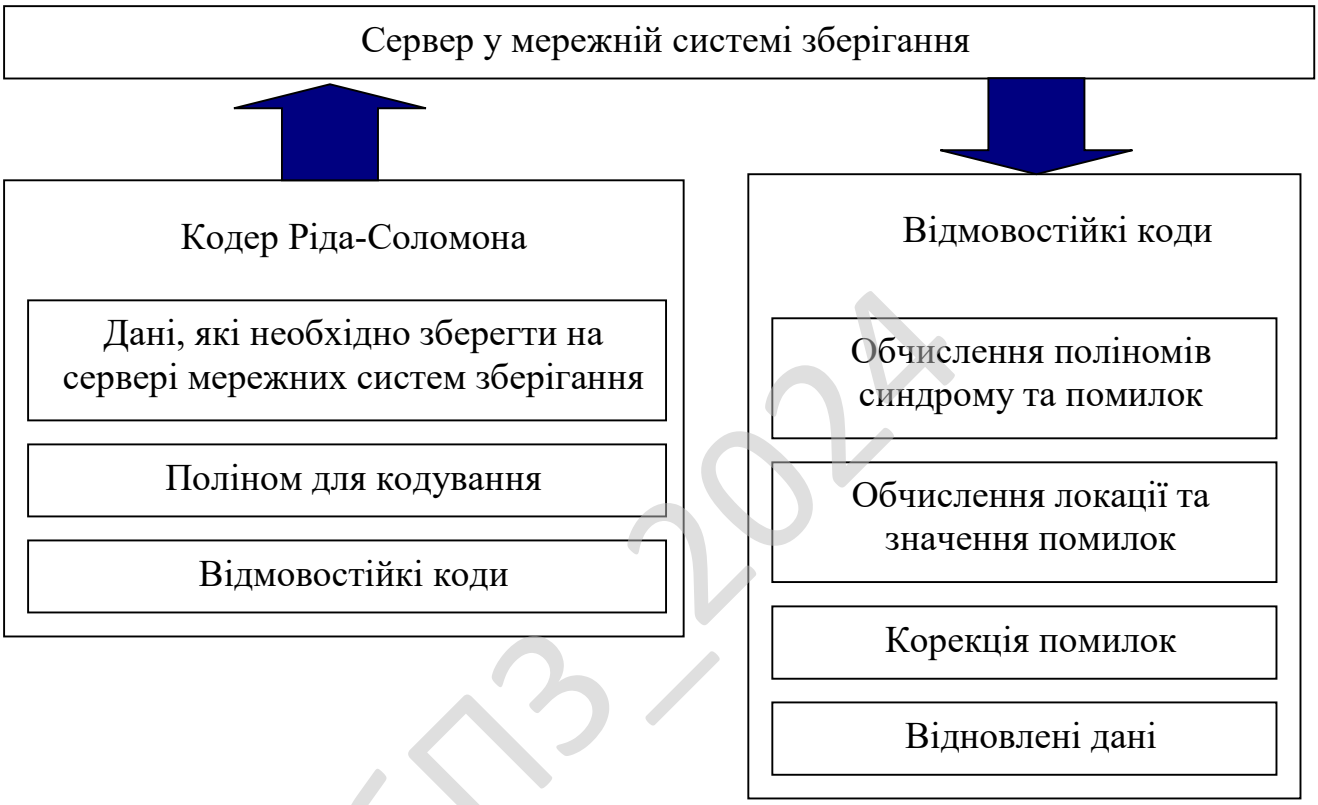

Рисунок 3.2 – Функціональна схема системи

Декодер Ріда-Соломона намагається визначити позицію й значення помилки для числа t помилок (або 2t втрат) і виправити помилки й втрати.

Обчислення синдрому

Обчислення синдрому схоже на обчислення парності. Кодове слово Ріда-Соломона має 2t синдромів, це залежить тільки від помилок (а не переданих кодових слів). Синдроми можуть бути обчислені шляхом підстановки 2t коріння утворюючого полінома g(x) в r(x).

Знаходження позицій символьних помилок

Це робиться шляхом рішення системи рівнянь із t невідомими. Існує кілька швидких алгоритмів для рішення цього завдання. Ці алгоритми використовують особливості структури матриці кодів Ріда-Соломона й сильно скорочують необхідну обчислювальну потужність. Робиться це у два етапи:

1. Визначення полінома локації помилок. Це може бути зроблене за допомогою алгоритму Berlekamp-Massey або алгоритму Евкліда. Алгоритм Евкліда використовується частіше на практиці, тому що його легше реалізувати, однак, алгоритм Berlekamp-Massey дозволяє одержати більш ефективну реалізацію встаткування й програм.

2. Знаходження кореня цього полінома. Це робиться із залученням алгоритму пошуку Chien.

Знаходження значень символьних помилок

Тут також потрібно вирішити систему рівнянь із t невідомими. Для рішення використовується швидкий алгоритм Forney.

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

Код Ріда-Соломона

Для реалізації перешкодостійкого зберіганням інформації у мережних дата-центрах, застосуємо алгоритм Ріда-Соломона.

Коди Ріда-Соломона – недвійкові циклічні коди, що дозволяють виправляти помилки в блоках даних. Елементами кодового вектора є не біти, а групи біт (блоки). Дуже поширені коди Ріда-Соломона, що працюють із байтами (октетами).

У цей час широко використовується в системах відновлення даних на різних носіях, у тому числі й у дата-центрах, при створенні архівів з інформацією для відновлення у випадку ушкоджень, у завадостійкому кодуванні.

Код Ріда-Соломона був винайдений в 1960 році співробітниками лабораторії Лінкольна Массачуссетского технологічного інституту Ірвином

Рідом і Густавом Соломоном. Ідея використання цього коду була представлена в статті «Polynomial Codes over Certain Finite Fields». Перше застосування код Ріда-Соломона одержав в 1982 році в серійному випуску компакт-дисків. Ефективний алгоритм декодування був запропонований в 1969 році Елвином Берлекемпом і Джеймсом Месси.

Коди Ріда-Соломона є важливим частковим випадком БЧХ-коду, корінь полінома, що породжує, якого лежать у тім же полі, над яким і будується код (m = 1).

Коди Боуза-Чоудхури-Хоквингхема (БЧХ коди) – у теорії кодування це широкий клас циклічних кодів, застосовуваних для захисту інформації від помилок. Відрізняється можливістю побудови коду із заздалегідь певними коригувальними властивостями, а саме, мінімальною кодовою відстанню.

Поліном, що породжує

Поліномом, що породжує, циклічного (n,k) коду C називається такий ненульовий $g(x) = \sum_{i=0}^{r} g_i x^i$ поліном з C, ступінь якого найменша й коефіцієнт при старшому ступені $g_r = 1$.

Теорема 3.1

Якщо C – циклічний (n,k) код і $g(x)$ – його поліном, що породжує, тоді ступінь $g(x)$ дорівнює $r = n - k$ і кожне кодове слово може бути єдиним чином представлено у вигляді $c(x) = m(x)g(x)$, де ступінь $m(x)$ менше або дорівнює $k - 1$.

Теорема 3.2

$g(x)$ – поліном, що породжує, циклічного (n,k) коду є дільником двочлена $x^n - 1$

Наслідок: у такий спосіб як поліном, що породжує, можна вибирати будь-який поліном, дільник $x^n - 1$. Ступінь обраного полінома буде визначати кількість перевірочних символів r, число інформаційних символів $k = n - r$.

Матриця, що породжує

Поліноми $g(x), xg(x), x^2g(x), \ldots, x^{k-1}g(x)$ лінійно незалежні, інакше

$m(x)g(x) = 0$ при ненульовому $m(x)$, що неможливо.

Значить кодові слова можна записувати, як і для лінійних кодів, наступним чином:

$$\overline{m}G = (m_0, m_1, ..., m_{k-1}) \begin{bmatrix} g(x) \\ xg(x) \\ ... \\ x^{k-1}g(x) \end{bmatrix} = m(x)g(x), \quad (3.1)$$

де G є матрицею, що породжує, $m(x)$ – інформаційним поліномом.

Матрицю G можна записати в символьній формі:

$$G = \begin{bmatrix} g_0 & g_1 & \dots & g_{r-1} & g_r & 0 & \dots & 0 \\ 0 & g_0 & \dots & g_{r-2} & g_{r-1} & g_r & \dots & 0 \\ . & . & \dots & . & . & . & \dots & . \\ 0 & 0 & \dots & 0 & g_0 & g_1 & \dots & g_r \end{bmatrix}$$

Перевірочна матриця

Для кожного кодового слова циклічного коду справедливо $c(x) = 0 \mod g(x)$. Тому перевірочну матрицю можна записати як $H = \begin{bmatrix} 1 & x & x^2 & \dots & x^{n-2} & x^{n-1} \end{bmatrix} \mod g(x)$.

Тоді:

$$\overline{c}H^T = \sum_{i=0}^{n-1} c_i x^i \bmod g(x). \quad (3.2)$$

Нехай α – елемент поля $GF(q)$ порядку n. Якщо α – *примітивний* елемент, то його порядок дорівнює q-*1*, т.е. $\alpha^{q-1}=1$, $\alpha^i \neq 1$, $0 < i < q - 1$.

Тоді нормований поліном $g(x)$ мінімального ступеня над полем $GF(q)$, корінням якого є d-1 ступенів, що йдуть підряд, $\alpha^{l_0}, \alpha^{l_0+1}, ..., \alpha^{i_0+d-1}$ елемента α, є поліномом, що породжує, коду над *полем* GF*(*q) $g(x) = (x - \alpha^{l_0})(x - \alpha^{l_0+1}) \dots (x - \alpha^{l_0+d-1})$:, де l_0 – деяке ціле число (у тому числі 0 і 1), за допомогою якого іноді вдається спростити кодер. Звичайно покладається $l_0 = 1$. Ступінь багаточлена $g(x)$ дорівнює $d - 1$.

Довжина отриманого коду n, мінімальна відстань d (мінімальна відстань d лінійного коду є мінімальним із всіх відстаней Хеммінга всіх пар кодових слів).

Код містить $r = d - 1 = \deg(g(x))$ перевірочний символ, де deg() позначає ступінь полінома; число інформаційних символів $k = n - r = n - d + 1$. У такий спосіб $d = n - k - 1$ і код Ріда-Соломона є роздільним кодом з максимальною відстанню (є оптимальним у змісті границі Синглтона).

Кодовий поліном $c(x)$ може бути отриманий з інформаційного полінома $m(x)$, $\deg m(x) \leq k - 1$, шляхом перемножування $m(x)$ і $g(x)$: $c(x) = m(x)g(x)$

Властивості

Код Ріда-Соломона над $GF(q^m)$, що виправляє t помилок, вимагає $2t$ перевірочних символів і з його допомогою виправляються довільні пакети довжиною t і менше. Відповідно до теореми про границю Рейгера, коди Ріда-Соломона є оптимальними з погляду співвідношення довжини пакета й можливості виправлення помилок – використовуючи $2t$ додаткових перевірочних символів виправляються t помилок (і менш).

Теорема (границя Рейгера). Кожний лінійний блоковий код, що виправляє всі пакети довжиною t і менш, повинен містити щонайменше $2t$ перевірочних символів.

Виправлення багаторазових помилок

Код Ріда-Соломона є одним з найбільш потужних кодів, що виправляють багаторазові пакети помилок. Застосовується в каналах, де пакети помилок можуть утворюватися настільки часто, що їх уже не можна виправляти за допомогою кодів, що виправляють одиночні помилки. $(q^m - 1, q^m - 1 - 2t)$-код Ріда-Соломона над полем $GF(q^m)$ з кодовою відстанню $d = 2t + 1$ можна розглядати як $((q^m - 1)m, (q^m - 1 - 2t)m)$-код над полем $GF(q)$, що може виправляти будь-яку комбінацію помилок, зосереджену в t або меншому числі блоків з m символів.

Найбільше число блоків довжини m, які може торкнути пакет довжини l_i, де $l_i <= mt_i - (m - 1)$, не перевершує t_i, тому код, що може виправити t блоків помилок, завжди може виправити й будь-яку комбінацію з p пакетів загальної довжини l, якщо $l + (m - 1) <= mt$.

Практична реалізація

Кодування за допомогою коду Ріда-Соломона може бути реалізовано двома способами: систематичним і несистематичним.

При несистематичному кодуванні інформаційне слово множиться на якийсь полином, що неприводиться, у полі Галуа. Отримане закодоване слово повністю відрізняється від вихідного й для добування інформаційного слова потрібно виконати операцію декодування й уже потім можна перевірити дані на зміст помилок. Таке кодування вимагає більші витрати ресурсів тільки на добування інформаційних даних, при цьому вони можуть бути без помилок.

При систематичному кодуванні до інформаційного блоку з k символів приписуються $2t$ перевірочних символів, при обчисленні кожного перевірочного символу використовуються всі k символів вихідного блоку.

У цьому випадку немає витрат ресурсів при добуванні вихідного блоку, якщо інформаційне слово не містить помилок, але кодер/декодер повинен виконати $k(n-k)$ операцій додавання й множення для генерації перевірочних символів. Крім того, тому що всі операції проводяться в поле Галуа, те самі операції кодування/декодування вимагають багато ресурсів і часу. Швидкий алгоритм декодування, заснований на швидкому перетворенні Фур'є, виконується за час порядку lnn^2.

Кодування

При операції кодування інформаційний поліном множиться на багаточлен, що породжує. Множення вихідного слова S довжини k на не приводиться полином, що, при систематичному кодуванні можна виконати в такий спосіб:

– До вихідного слова приписуються $2t$ нулів, виходить поліном $T = Sx^{2t}$.

– Цей поліном ділиться на поліном, що породжує, G, перебуває залишок R, $Sx^{2t} = QG + R$, де Q – частка.

– Цей залишок й буде коригувальним кодом Ріда-Соломона, він приписується до вихідного блоку символів. Отримане кодове слово $C = Sx^{2t} + R$.

Кодер будується зі регістрів зсуву, суматорів і перемножувачів. Регістр

зсуву складається з комірок пам'яті, у кожній з яких перебуває один елемент поля Галуа.

Наведений як приклад кодер Ріда-Соломона генерує 16 коригувальних байт, що дозволяє виправляти до 8 і виявляти до 16 помилок у кадрі даних.

Перемножувачі на константи *GF*(0)...*GF*(15) у полі Галуа реалізуються в такий спосіб: спочатку вихідне число й константа перетворяться в індексну форму, потім складаються в межах байта без обліку переносу.

Результатом операції є результат додавання, перетворена обернено в поліноміальну форму.

При переході від однієї форми подання даних до інший доцільно використовувати таблицю істинності розміром 256 байт, що становить ємність одного EAB (Embedded Array Block – блок зосередженої пам'яті). Для реалізації кодеру потрібно 16 таких перемножувачів, при цьому те саме число множиться на різні константи, що дозволяє використовувати для його перекладу в індексну форму один EAB.

Для перекладу результатів у поліноміальну форму потрібно вже 16 таких таблиць, що вимагає застосування IMC FPGA дуже великої ємності. У запропонованій схемі використовується тактування кодера із частотою, в 8 разів перевищуючу частоту надходження байт даних.

Це дає можливість використовувати дві пари «суматор – EAB", мультиплексуючи константи на входах суматорів і дозволяючи роботу регістрів-накопичувачів у моменти появи відповідних даних на виходах засувок EAB.

Символу «C» відповідають дві константи *GF*(*n*).

Символ «L» у логіці регістрів-накопичувачів відповідає наступний: вихід компаратора нуля (символи CMP0 і SYNC) дозволяє роботу схеми «АБО що виключає ", на входи якої подаються вихід попереднього регістра й EAB. Якщо ж вектор зворотного зв'язку дорівнює "0", схема пропускає дані з виходу попереднього регістра-накопичувача на вхід наступного.

У результаті кодер з урахуванням схеми синхронізації займає 255 LE (Logic Element – логічний елемент) і 3 EAB, що дозволяє розмістити його в IMC

EPF10K10. Після оптимізації розміщення схеми на кристалі FPGA швидкодія схеми досяглася 11,57 Мгц (частота надходження байт даних, далі – байтова частота).

При використанні IMC EPF10K20, у складі якої 6 EAB, використовуючи 4 пари "суматор – EAB", можна тактувати кодер із частотами, що перевищують байтову частоту не в 8, а в 4 рази, що дозволить підняти її до 25...30 Мгц.

Декодування

Декодер, що працює по авторегресивному спектральному методі декодування, послідовно виконує наступні дії:

– Обчислює синдром помилки.

– Будує поліном помилки.

– Знаходить корінь даного полінома.

– Визначає характер помилки.

– Виправляє помилки.

Обчислення синдрому помилки

Обчислення синдрому помилки виконується синдромним декодером, що ділить кодове слово на багаточлен, що породжує. Якщо при діленні виникає остача, то в слові є помилка. Остача від ділення є синдромом помилки.

Побудова полінома помилки

Обчислений синдром помилки не вказує на положення помилок. Ступінь полінома синдрому дорівнює 2*t*, що багато менше ступеня кодового слова *n*. Для одержання відповідності між помилкою і її положенням у повідомленні будується поліном помилок.

Поліном помилок реалізується за допомогою алгоритму Берлекемпа-Месси, або за допомогою алгоритму Евкліда. Алгоритм Евкліда має просту реалізацію, але вимагає більших витрат ресурсів. Тому частіше застосовується більше складний, але менш затратоємний алгоритм Берлекемпа-Месси. Коефіцієнти знайденого полінома безпосередньо відповідають коефіцієнтам помилкових символів у кодовому слові.

Алгоритм Евкліда виконується наступним чином.

Нехай a і b суть цілі числа, не рівні одночасно нулю, і послідовність чисел $a, b, r_1 > r_2 > r_3 > r_4 > \cdots > r_n$ визначена тим, що кожне r_k це остача від ділення перед-попереднього числа на попереднє, а передостаннє ділиться на останнє націло, тобто

$$\begin{aligned} a &= bq_0 + r_1 \\ b &= r_1q_1 + r_2 \\ r_1 &= r_2q_2 + r_3 \\ &\cdots \\ r_{n-1} &= r_nq_n \end{aligned} \qquad (3.3)$$

Тоді (a,b), найбільший загальний дільник a і b, дорівнює r_n, останньому ненульовому члену цієї послідовності.

Існування таких $r_1,r_2,...$, тобто можливість ділення з остачею m на n для будь-якого цілого m і цілого $n \neq 0$, доводиться індукцією по m.

Коректність цього алгоритму випливає з наступних двох тверджень:

– Нехай $a = bq + r$, тоді $(a,b) = (b,r)$.

– $(0,r) = r$. для будь-якого ненульового r.

Знаходження корня

На цьому етапі шукаються коріння полінома помилки, що визначають положення перекручених символів у кодовому слові. Реалізується за допомогою процедури Ченя, рівносильній повному перебору. У поліном помилок послідовно підставляються всі можливі значення, коли поліном звертається в нуль – коріння знайдені.

Визначення характеру помилки і її виправлення

По синдрому помилки й знайдених корінь полінома за допомогою алгоритму Форни визначається характер помилки й будується маска перекручених символів. Ця маска накладається на кодове слово за допомогою операції XOR і перекручені символи відновлюються. Після цього відкидаються перевірочні символи й виходить відновлене інформаційне слово.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання магістерської дипломної роботи, наведена на рисунку 3.3.

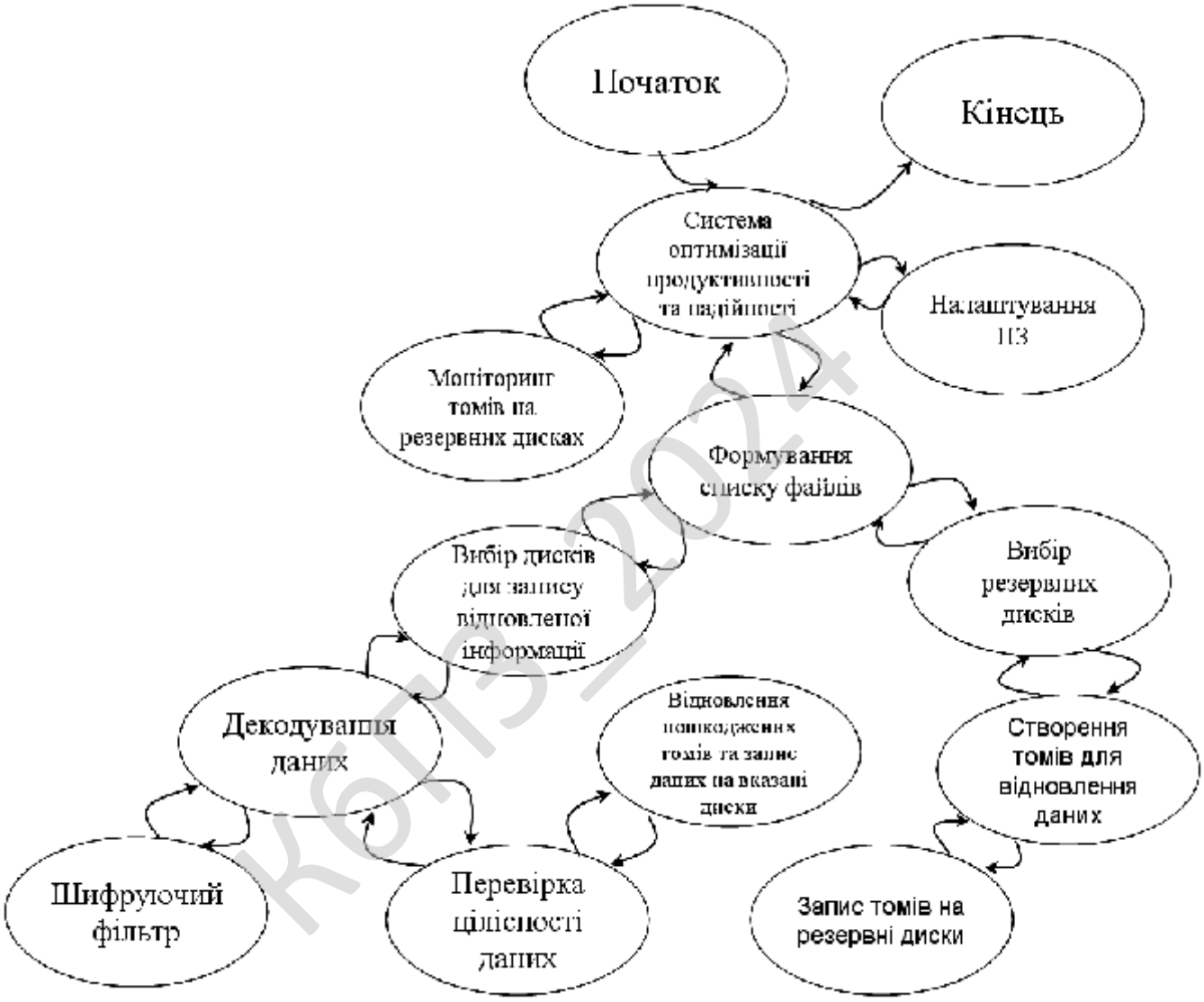

Рисунок 3.3 – Діаграма взаємодії процесів

При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

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

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Під час роботи над магістерською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято

позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

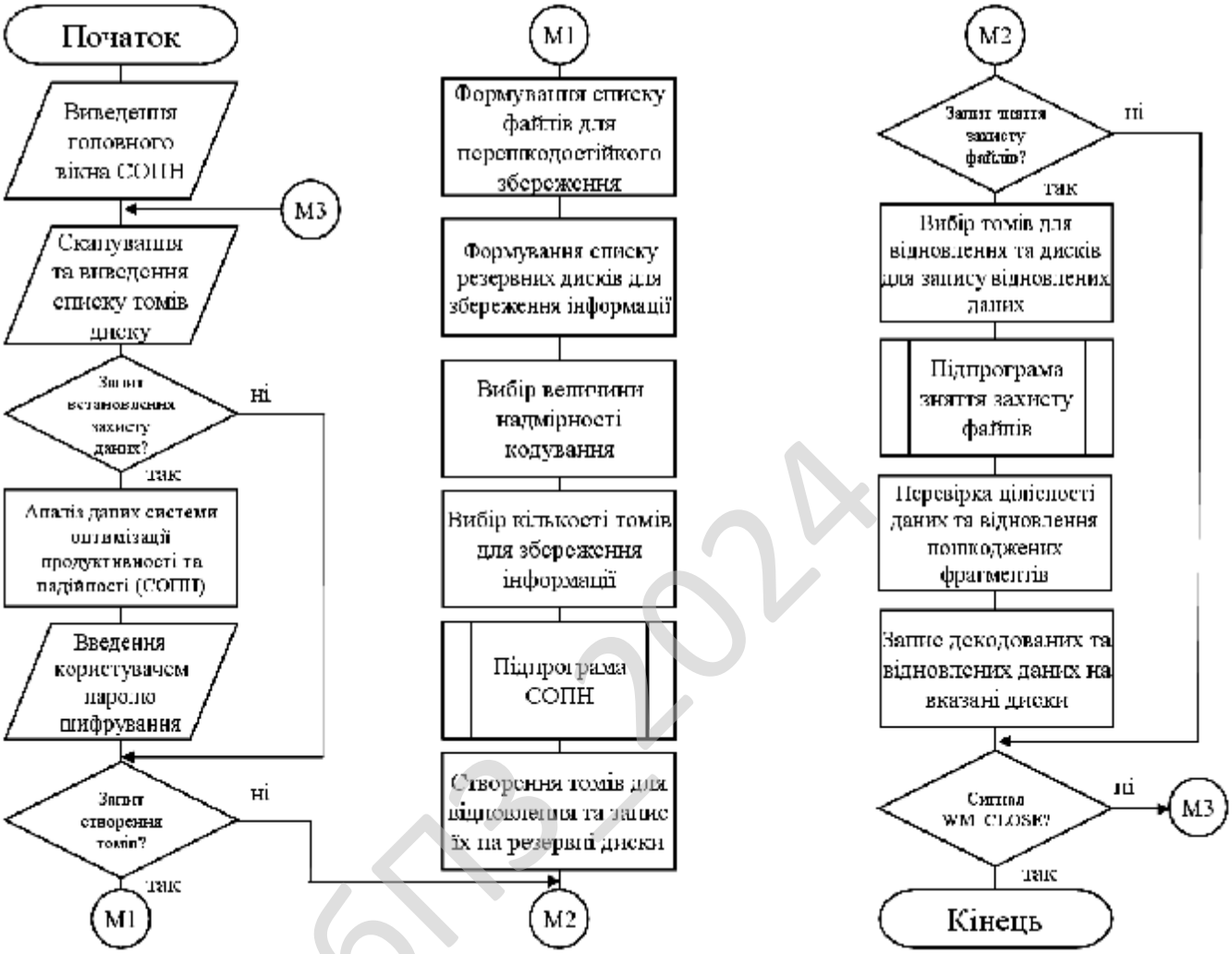

Рисунок 4.1 – Блок-схема основної програми

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

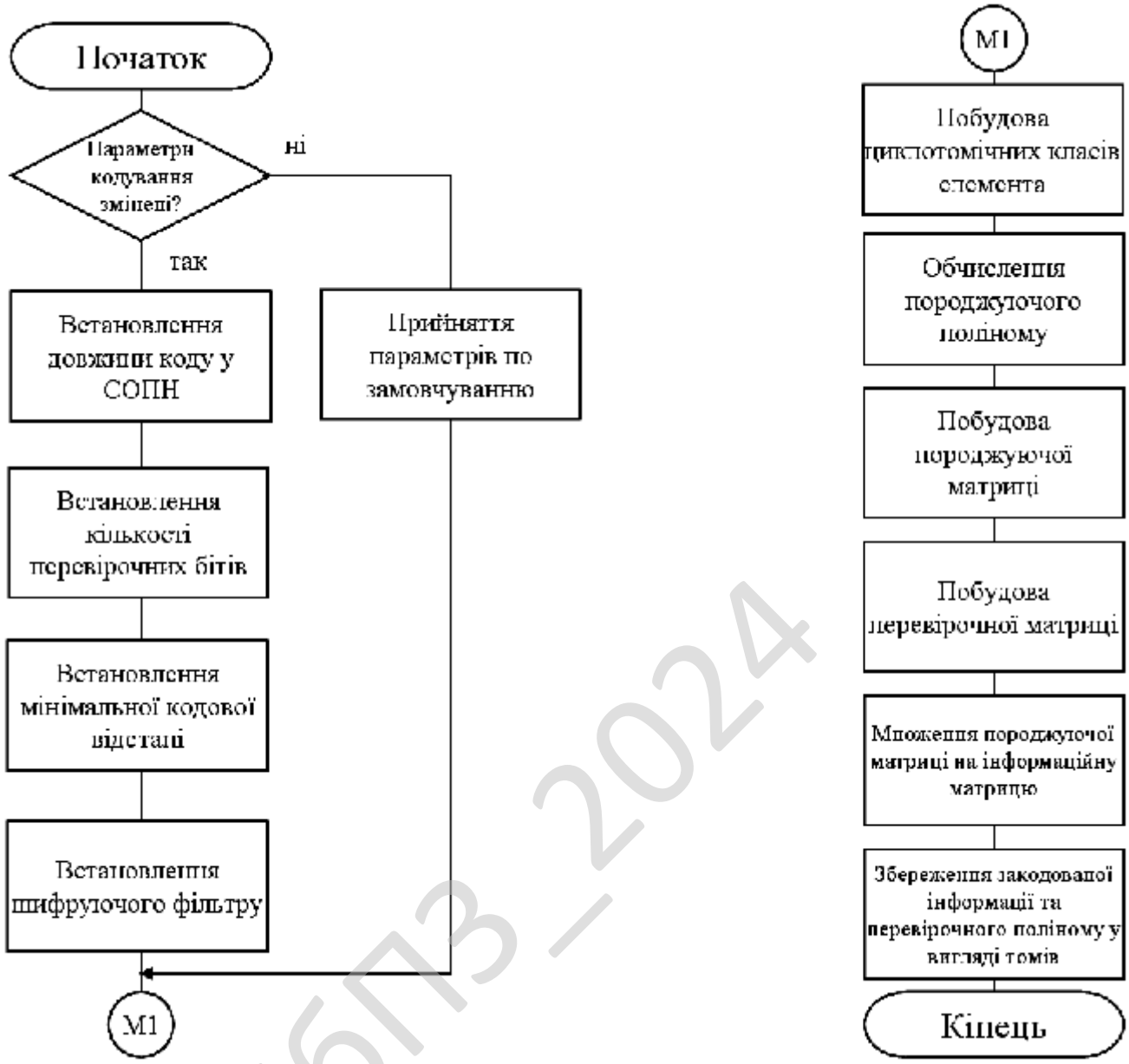

Рисунок 4.2 – Блок-схема роботи підпрограми

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Також при розробці магістерської дипломної роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу); Діаграма класів.

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилатися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати

діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

Опис алгоритмів функціонування системи

Коди Ріда – Соломона (Reed-Solomon codes) – недвійкові циклічні коди, що дозволяють виправляти помилки в блоках даних. Елементами кодового вектора є не біти, а групи бітів (блоки). Дуже поширені коди Ріда – Соломона, що працюють з байтами (октет).

Код Ріда – Соломона є окремим випадком БЧХ-коду. Коди Боуза – Чоудхурі – Хоквінгема (БЧХ-коди) – в теорії кодування це широкий клас

циклічних кодів, що застосовуються для захисту інформації від помилок (попередня корекція помилок).

Відрізняється можливістю побудови коду із заздалегідь визначеними коригувальними властивостями, а саме, мінімальною кодовою відстанню. Окремим випадком БЧХ-кодів є Код Ріда-Соломона. Код винайшов в 1959 році А.Хоквінгем (Hocquenghem), і незалежно в 1960 році Р.Боуз (Bose) і Д.Рой-Чоудхурі (Ray-Chaudhuri). Код отримав свою назву (BCH code) від прізвищ їх авторів.

Коди БЧХ є узагальненням кодів Хеммінга і дозволяють виправляти кратні помилки.

В даний час широко використовується в системах відновлення даних з компакт-дисків, при створенні архівів з інформацією для відновлення у випадку ушкоджень, в завадостійкому кодуванні.

Код Ріда – Соломона використовується при запису і зчитуванні в контролерах оперативної пам'яті, при архівуванні даних, запису інформації на жорсткі диски (ECC-пам'ять), запису на CD/DVD диски.

Навіть якщо пошкоджено значний обсяг інформації, зіпсовано кілька секторів дискового носія, то коди Ріда – Соломона дозволяють відновити велику частину втраченої інформації.

Також використовується при запису на такі носії, як магнітні стрічки і штрихкоди. Цей алгоритм кодування використовується при передачі даних по мережах WiMAX, в оптичних лініях зв'язку, у супутниковому та радіорелейному зв'язку.

Метод прямої корекції помилок в трафіку (Forward Error Correction, FEC) ґрунтується на кодах Ріда – Соломона.

Кодуємі дані передаються через масив data[i], де i = 0...(k – 1), а згенеровані символи парності заносяться в масив b[0]...b[2 * t – 1].

Вихідні й результуючі дані повинні бути представлені в поліноміальної формі (тобто у звичайній формі машинного подання даних).

Кодування виробляється з використанням сдвигового feedback-регістра, заповненого відповідними елементами масиву g[] з породженим поліномом.

```
e_rs()
{
       int i, j;
       int feedback;
// ініціалізація поля біт парності нулями
       for (i = 0; i < n - k; i++) b[i] = 0;
// обробляємо всі символи
// вихідних даних праворуч ліворуч
       for (i = k - 1; i >= 0; i-i--)
       {
// готовимо (data[i] + b[n - k - 1]) до множення на g[i]
// тобто складаємо черговий "захоплений" символ вихідних
// даних з молодшим символом біт парності (відповідного
// "регістру" b2 t-1, і переводимо його в
// індексну форму, зберігаючи результат у регістрі feedback;
// як ми вже говорили, сума двох індексів є добуток  поліномів
              feedback = index_of[data[i] ^ b[n - k - 1]];
 // є ще символи для обробки?
              if (feedback != -1)
              {
 // здійснюємо зрушення ланцюга bx-регістрів
                     for (j = n - k - 1; j > 0; j-j--)
  // якщо поточний коефіцієнт g - це дійсний
 // (тобто ненульовий коефіцієнт, те
  // множимо feedback на відповідний g-коефіцієнт
  // і складаємо його з наступних елементів ланцюжка
if (g[j] != -1) b[j] = b[j - 1] ^ alpha_to[(g[j] + feedback) % n];
                     else
   // якщо поточний коефіцієнт g - це нульовий коефіцієнт,
   // виконуємо одне лише зрушення без множення, переміщаючи
   // символ з одного m-регістра в інший
                            b[j] = b[j - 1];
   // закільцьовуємо вихідний символ у крайній лівий b 0-регістр
              b[0] = alpha_to[(g[0] + feedback) % n];
       }
       else
       {
  // розподіл завершений,
  // здійснюємо останнє зрушення регістра,
```

```
  // на виході регістра буде частка, що губиться,
  // а в самому регістрі – шуканий залишок
                    for (j = n – k – 1; j > 0; j-j--) b[j] = b[j – 1]; b[0] = 0;
              }
       }
}
```

Нижче наведемо вихідний текст підпрограми декодера Ріда-Соломона. Процедура декодування кодів Ріда-Соломона складається з декількох кроків. Спочатку ми обчислюємо 2 t-символьний синдром шляхом постановки alpha**i в recd(x), де recd – отримане кодове слово, попередньо переведене в індексну форму. По факті обчислення recd(x) ми записуємо черговий символ синдрому в s[i], де i приймає значення від 1 до 2t, залишаючи s[0] рівним нулю.

Потім, використовуючи ітеративний алгоритм Берлекэмпа (Berlekamp), ми знаходимо поліном локатора помилки – elp[i].

Якщо ступінь elp перевищує собою величину t, ми неспроможні скорегувати всі помилки й обмежуємося виводом повідомлення про непереборну помилку, після чого робимо аварійний вихід з декодера. Якщо ж ступінь elp не перевищує t, ми підставляємо alpha**i, де i = 1...n в elp для обчислення корінь полінома. Обіг знайдений корінь дає нам позиції перекручених символів.

Якщо кількість певних позицій перекручених символів менше ступеня elp, перекручуванню піддалося більш ніж t символів і ми не можемо відновити їх.

У всіх інших випадках відновлення оригінального вмісту перекручених символів цілком можливо.

У випадку, коли кількість помилок свідомо велико, для їхнього виправлення декодуємі символи проходять крізь декодер без яких або змін.

```
decode_rs()
{
       int i, j, u, q;
       int s[n – k + 1];
// поліном синдрому помилки
       int elp[n – k + 2][n – k];
// поліном локатора помилки лямда
       int d[n – k + 2];
       int l[n – k + 2];
```

```
        int u_lu[n - k + 2],
int count = 0, syn_error = 0, root[t], loc[t], z[t + 1], err[n], reg[t + 1];
// переводимо отримане кодове слово в індексну форму
// для спрощення обчислень
        for (i = 0; i < n; i++) recd[i] = index_of[recd[i]];
        // обчислюємо синдром
        //---------------------------------
        for (i = 1; i <= n - k; i++)
        {
                s[i] = 0;        // ініціалізація s-регістра
                                 // на його вхід за замовчуванням надходить нуль
// виконуємо s[i] += recd[j] * ij
// тобто беремо черговий символ декодуємих даних,
// множимо його на порядковий номер даного символу,
// помножений на номер чергового оберту й складаємо
// отриманий результат із умістом s-регістра;
// по факті вичерпання всіх декодуємих символ ми
// повторюємо весь цикл обчислень знову - по одному
// разу для кожного символу парності
for(j=0; j < n; j++) if (recd[j] != -1) s[i] ^= alpha_to[(recd[j] + i * j) % n];
                if (s[i] != 0) syn_error = 1;
// якщо синдром не дорівнює нулю, зводимо прапор помилки
// перетворимо синдром з поліноміальної форми в індексну
                s[i] = index_of[s[i]];
        }
// корекція помилок
// якщо є помилки, намагаємося їх скорегувати
        if (syn_error)
        {
 // обчислення полінома локатора ламбла
// обчислюємо поліном локатора помилки через ітеративний алгоритм
// Берлекемпа. Випливаючи термінології Lin and Costello (див. "Error
// Control Coding: Fundamentals and Applications" Prentice Hall 1983
// ISBN 013283796) d[u] являє собою m ("мю"), що виражає
// розбіжність (discrepancy), де u = m + 1 і m є номер кроку
// з діапазону від -1 до 2t. У Блейхута та ж сама величина
// позначається D(x) ("дельта") і називається нев'язання.
// l[u] являє собою ступінь elp для даного кроку ітерації,
// u_l[u] являє собою різницю між номером кроку й ступенем elp

// ініціалізація елементи таблиці
                d[0] = 0;
```

```
// індексна форма
              d[1] = s[1];
// індексна форма
              elp[0][0] = 0;
// індексна форма
              elp[1][0] = 1;
// поліноміальна форма
              for (i = 1; i < n - k; i++)
              {
                     elp[0][i] = -1;
// індексна форма
                     elp[1][i] = 0;
// поліноміальна форма
}
l[0] = 0; l[1] = 0; u_lu[0] = -1; u_lu[1] = 0; u = 0;
do
{
u++;
if (d[u] == -1)
{
       l[u + 1] = l[u];
       for (i = 0; i <= l[u]; i++)
       {
              elp[u + 1][i] = elp[u][i];
              elp[u][i] = index_of[elp[u][i]];
       }
}
       else
{
// пошук слів з найбільшим u_lu[q], таких що d[q] != 0
       q = u - 1;
       while ((d[q] == -1) && (q > 0)) q-q--;
// знайдений перший ненульовий d[q]
       if (q > 0)
       {
              j = q;
              do
       {
              j-j-- ;
              if ((d[j] != -1) && (u_lu[q] < u_lu[j]))
              q = j;
       } while (j > 0);
```

```
};
 // як тільки ми знайдемо q, такий що d[u] !=0
// і u_lu[q] є максимум
// запишемо ступінь нового elp полінома
if (l[u] > l[q] + u - q) l[u + 1] = l[u]; else l[u + 1] = l[q] + u - q;
// формуємо новий elp(x)
              for (i = 0; i < n - k; i++) elp[u + 1][i] = 0;
                 for (i = 0; i <= l[q]; i++)
                   if (elp[q][i] != -1)
elp[u + 1][i + u - q] = alpha_to[(d[u] + n - d[q] + elp[q][i]) % n];
                     for (i = 0; i <= l[u]; i++)
                     {
                            elp[u + 1][i] ^= elp[u][i];
                            // перетворимо старий elp
                            // в індексну форму
                            elp[u][i] = index_of[elp[u][i]];
                     }
              }
              u_lu[u + 1] = u - l[u + 1];
//---------------------------
              if (u < n - k)          // на останній ітерації розбіжність
              {                       // не було виявлено
             if (s[u + 1] != -1) d[u + 1] = alpha_to[s[u + 1]]; else d[u + 1] = 0;
                     for (i = 1; i <= l[u + 1]; i++)
                            if ((s[u + 1 - i] != -1) && (elp[u + 1][i] != 0))
                            d[u + 1] ^= alpha_to[(s[u + 1 - i] + index_of[elp[u +
1][i]]) % n];
                     // переводимо d[u + 1] в індексну форму
                     d[u + 1] = index_of[d[u + 1]];
              }
       } while ((u < n - k) && (l[u + 1] <= t));
       // розрахунок локатора завершений
       //------------------------
       u++ ;
       if (l[u] <= t)
       {
              // корекція помилок можлива
              // переводимо elp в індексну форму
              for (i = 0; i <= l[u]; i++) elp[u][i] = index_of[elp[u][i]];
// знаходження корінь полінома локатора помилки
//---------------------------
   for (i = 1; i <= l[u]; i++) reg[i] = elp[u][i]; count = 0;
```

```
   for (i = 1; i <= n; i++)
   {
          q = 1 ;
          for (j = 1; j <= l[u]; j++)
          if (reg[j] != -1)
          {
                 reg[j] = (reg[j] + j) % n;
                 q ^= alpha_to[reg[j]];
          }
          if (!q)
          { // записуємо корінь і індекс позиції помилки
                 root[count] = i;
                 loc[count] = n – i;
                 count++;
          }
   }
   if (count == l[u])
   { // немає корінь – ступінь elp < t помилок
          // формуємо поліном z(x)
          for (i = 1; i <= l[u]; i++) // Z[0] завжди дорівнює 1
          {
                 if ((s[i] != -1) && (elp[u][i] != -1))
                        z[i] = alpha_to[s[i]] ^ alpha_to[elp[u][i]];
                 else
                        if ((s[i]! =- 1) && (elp[u][i] == -1))
                               z[i] = alpha_to[s[i]];
                        else
                               if ((s[i]==-1) && (elp[u][i]!=-1))
                                      z[i] = alpha_to[elp[u][i]];
                               else
                                             z[i] = 0 ;
          for (j = 1; j < i; j++)
                 if ((s[j] != -1) && (elp[u][i – j] != -1))
                        z[i] ^= alpha_to[(elp[u][i – j] + s[j]) % n];
          // переводимо z[i] в індексну форму
          z[i] = index_of[z[i]];
   }
// обчислення значення помилок у позиціях loc[i]
//---------------------------
 for (i = 0; i < n; i++)
 {
        err[i] = 0;
```

```
   // переводимо recd[] у поліноміальну форму
    if (recd[i] != -1) recd[i] = alpha_to[recd[i]]; else recd[i] = 0;
 }
 // спочатку обчислюємо чисельник помилки
 for (i = 0; i < l[u]; i++)
 {
        err[loc[i]] = 1;
        for (j = 1; j <= l[u]; j++)
               if (z[j] != -1)
                  err[loc[i]] ^= alpha_to[(z[j] + j * root[i]) % n];
        if (err[loc[i]] != 0)
        {
        err[loc[i]] = index_of[err[loc[i]]];
q = 0 ; // формуємо знаменник коефіцієнта помилки
for (j = 0; j < l[u]; j++)
if (j != i) q += index_of[1 ^ alpha_to[(loc[j] + root[i]) % n]];
q = q % n; err[loc[i]] = alpha_to[(err[loc[i]] - q + n) % n];
// recd[i] повинен бути в поліноміальній формі
recd[loc[i]] ^= err[loc[i]];
}
 }
}
else // немає корінь,
// рішення системи рівнянь неможливо, тому що ступінь elp >= t
{
               // переводимо recd[] у поліноміальну форму
               for (i = 0; i < n; i++)
                     if (recd[i] != -1) recd[i] = alpha_to[recd[i]];
               else
                     recd[i] = 0; // виводимо інформаційне слово як є
       }
               else // ступінь elp > t, рішення неможливо
       {
// переводимо recd[] у поліноміальну форму
       for (i = 0; i < n; i++)
               if (recd[i] != -1)
                     recd[i] = alpha_to[recd[i]];
               else
                     recd[i] = 0 ;
// виводимо інформаційне слово як є
       }
               Else
```

```
// помилок не виявлено
        for (i = 0; i < n; i++) if (recd[i] != -1) recd[i] = alpha_to[recd[i]];
else recd[i] = 0;
}
```

4.2 Захист розробленого програмного забезпечення

Tiny Encryption Algorithm (TEA) [1] – блочний алгоритм шифрування типу «Мережі Фейстеля». Алгоритм був розроблений на факультеті комп'ютерних наук Кембриджського університету Девідом Вілером (David Wheeler) і Роджером Нідгемом (Roger Needham) та вперше представлений в 1994 році [2] на симпозіумі зі швидкими алгоритмами шифрування в Льовені (Бельгія).

Шифр не патентований, широко використовується в ряді криптографічних додатків і широкому спектрі апаратного забезпечення, завдяки вкрай низькими вимогами до пам'яті й простоті реалізації. Алгоритм має як програмну реалізацію на різних мовах програмування, так і апаратну реалізацію на інтегральних схемах типу FPGA.

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму TEA, який заснований на бітових операцій з 64-бітним блоком, має 128-бітний ключ шифрування. Стандартна кількість раундів мережі Фейстеля біля 64 (32 циклу), однак, для досягнення найкращої продуктивності або шифрування, число циклів можна варіювати від 8 (16 раундів) до 64 (128 раундів). Мережа Фейстеля несиметрична через використання в якості операції накладення додавання за модулем 2^{32}.

Перевагами шифру є його простота в реалізації, невеликий розмір коду й досить висока швидкість виконання, а також можливість оптимізації виконання на стандартних 32-бітних процесорах, так як в якості основних операцій використовуються операції виключна «АБО» (XOR), побітового зсуву й додавання за модулем 2^{32}. Оскільки алгоритм не використовує таблиць підстановки і раундова функція досить проста, алгоритму потрібно не менше 16

циклів (32 раундів) для досягнення ефективної дифузії, хоча повна дифузія досягається вже через 6 циклів (12 раундів).

Алгоритм має відмінну стійкість до лінійного криптоаналізу і досить гарну до диференціального криптоаналізу. Головним недоліком цього алгоритму шифрування є його вразливість до атак «на пов'язаних ключах» (англ. Related-key attack). Через простий розклад ключів кожен ключ має 3 еквівалентних ключа. Це означає, що ефективна довжина ключа складає всього 126 біт [3] [4], тому даний алгоритм не слід використовувати в якості геш-функції.

Опис алгоритму

Вихідний текст розбивається на блоки по 64 біта кожен. 128-бітний ключ K ділиться на чотири 32-бітних підключа K [0], K [1], K [2] і K [3]. На цьому підготовчий процес закінчується, після чого кожен 64-бітний блок шифрується протягом 32 циклів (64 раундів) за нижченаведеним алгоритмом. [5]

Припустимо, що на вхід n-го раунду ($1 \leq n \leq 64$) надходять права й ліва частини (L_n, R_n), тоді на виході n-го раунду будуть ліва й права частини (L_{n+1}, R_{n+1}), які обчислюються за такими правилами:

$$L_{n+1} = R_n.$$

Якщо $n = 2 * i - 1$ для $1 \leq i \leq 32$ (непарні раунди), то:

$$R_{n+1} = L_n (\{ [R_n 4] K [0] \} \{ R_n i * \delta \} \{ [R_n 5] K [1] \}).$$

Якщо $n = 2 * i$ для $1 \leq i \leq 32$ (парні раунди), то:

$$R_{n+1} = L_n (\{ [R_n 4] K [2] \} \{ R_n i * \delta \} \{ [R_n 5] K [3] \}).$$

Де

X Y – операція додавання чисел X і Y за модулем 232.

X Y – побітове виключне АБО» (XOR) чисел X і Y, яке в мові програмування Сі позначається як X ^ Y

X Y і X Y – операції побітового зсуву числа X на Y біт вліво й вправо відповідно.

Константа δ була виведена з Золотого перерізу:

$$\delta = (-1) * 2^{31} = 2654435769 = 9E3779B9_h.$$

У кожному раунді константа множиться на номер циклу i. Це було зроблено для запобігання простих атак, заснованих на симетрії раундів.

Також очевидно, що в алгоритмі шифрування TEA немає як такого алгоритму розкладу ключів. Замість цього в непарних раундах використовуються підключі K [0] та [1], у парних – K [2] і [3].

Так як це блочний шифроалгоритм, де довжина блоку 64-біт, а довжина даних може бути не кратна 64-біт, значення всіх байтів, які доповнюють блок до кратності в 64-біт, встановлюється в 0x01.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання магістерської дипломної роботи.

Розроблене програмне забезпечення системи оптимізації продуктивності та надійності мережних систем зберігання Big Data складається з наступних функціональних блоків:

– Навігаційне меню: Додатки; Налаштування системи оптимізації; Фільтри; Довідка.

– Розділи: Файли; резервні копії даних; Дії.

– Вікна обрання групи та функціональні кнопки ПЗ.

– Вікно виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

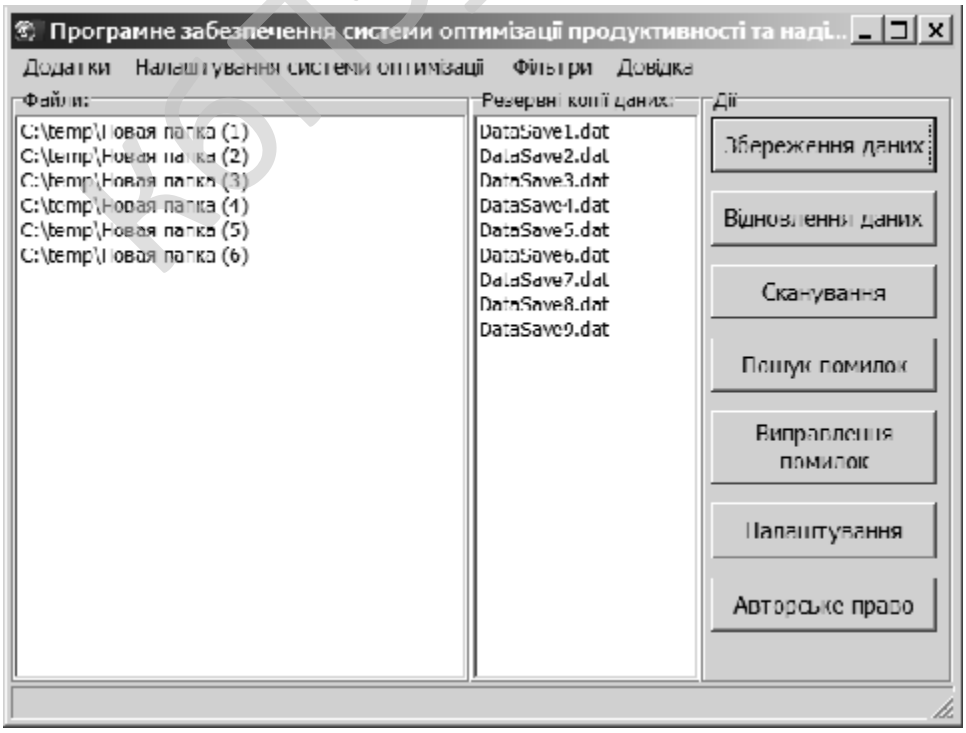

Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

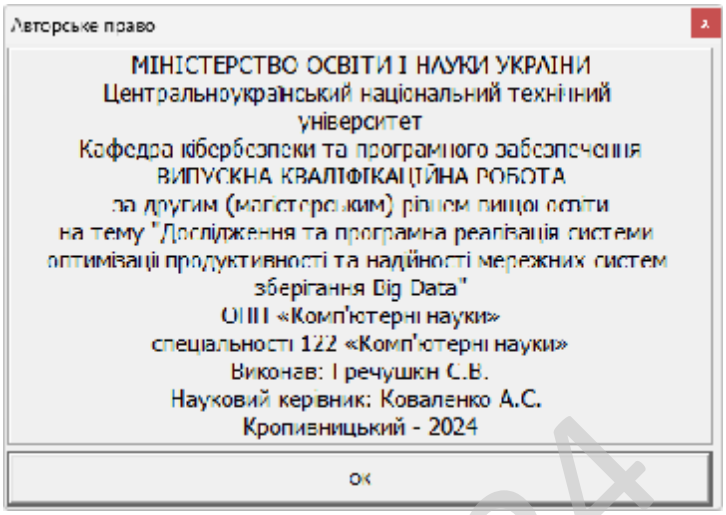

Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом білої скриньки та чорної скриньки.

Тестування форматом білої скриньки засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Проводилось тестування чорної скриньки.

Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ІТс).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його ОТ).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ІТс;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору ОТ.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій;

– Помилок інтерфейсу;

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;

– Помилок характеристик (необхідна ємність пам'яті і т.д.);

– Помилок ініціалізації та завершення.

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 (не повнофункціональна 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 (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості. Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму. В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Метою розробки є дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Об'єктом дослідження є процес оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Предметом дослідження є методи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Методи дослідження базуються на методах Big Data та теорії надійності, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод оптимізації продуктивності та надійності мережних систем зберігання Big Data.

– Розроблено вітчизняний продукт оптимізації продуктивності та надійності мережних систем зберігання Big Data, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та програмної реалізації системи оптимізації продуктивності та надійності мережних систем зберігання Big Data можуть бути цікаві різним групам зацікавлених осіб. Основні групи перераховано на рисунку 7.1.

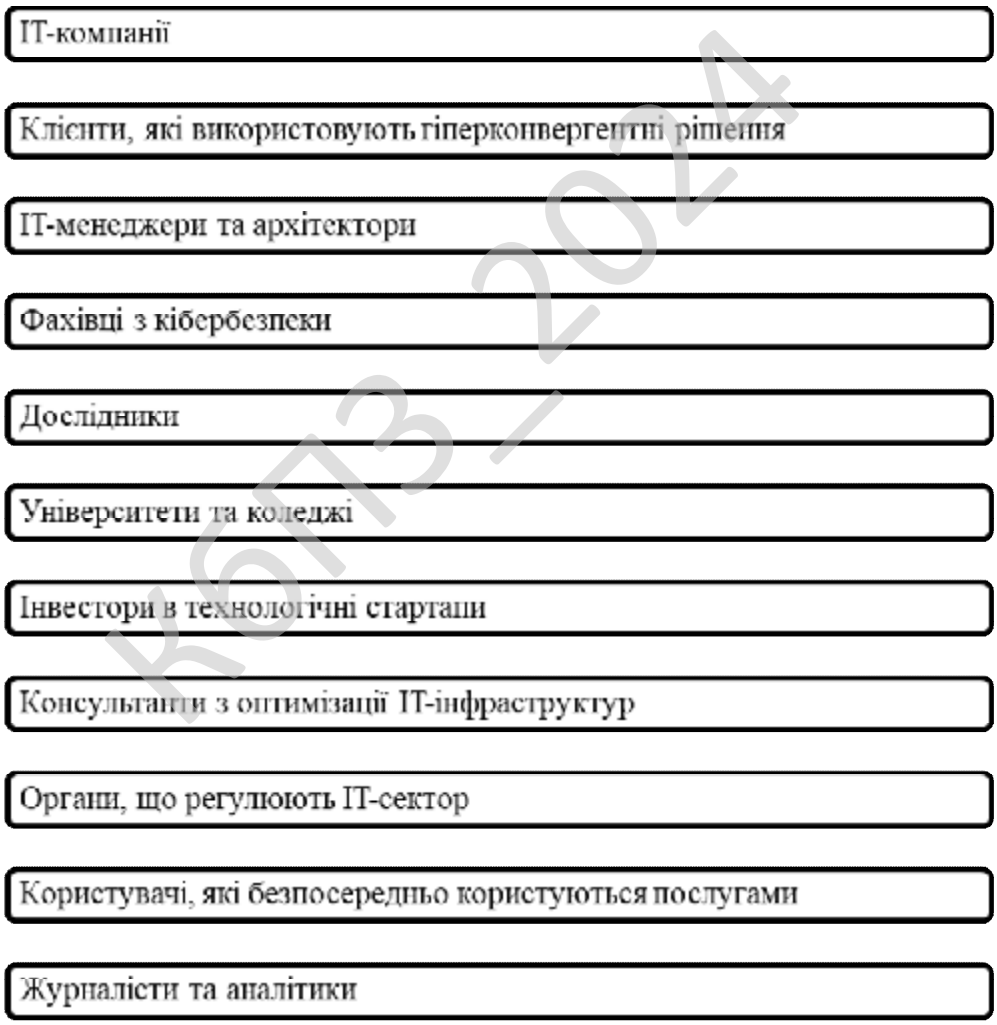

Рисунок 7.1 – Цільова аудиторія

Таким чином, результати дослідження та програмної реалізації системи оптимізації продуктивності та надійності мережних систем зберігання Big Data можуть зацікавити широкий спектр учасників ринку, що може сприяти розвитку технологій і впровадженню нових рішень в бізнесі.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Оцінка привабливості програмної реалізації системи оптимізації продуктивності та надійності мережних систем зберігання Big Data може бути здійснена за допомогою методів експертних оцінок.

Спочатку слід визначити критерії, які будуть використані для оцінки привабливості проекту. Це можуть бути критерії окреслені на рисунку 7.2.

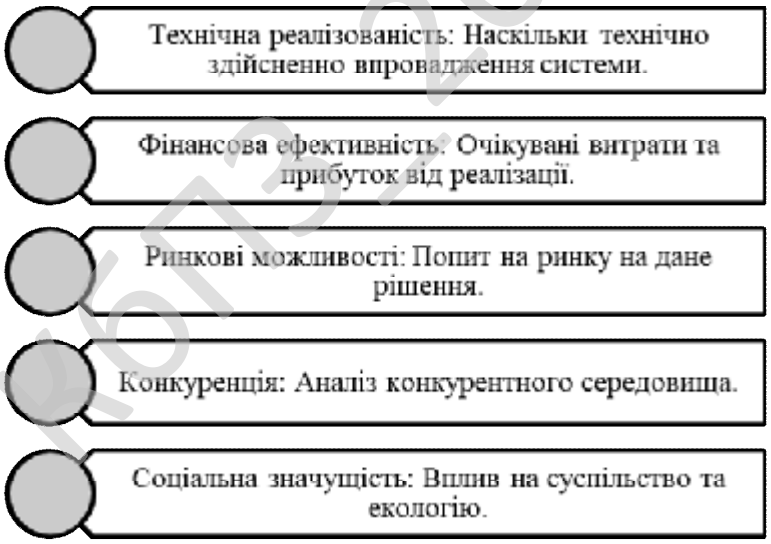

Рисунок 7.2 – Критерії експертної оцінки

Сформуємо групу експертів, які мають досвід у відповідних галузях (IT, бізнес, фінанси, маркетинг). Це: IT-архітектори, бізнес-аналітики, спеціалісти з продажу, фінансові аналітики. Експертам пропонується оцінити кожен критерій за шкалою, наприклад, від 1 до 5, де 1 – низька привабливість, а 5 – висока. Після

цього результати підраховуються і виводиться середній по критерію: технічна реалізованість: 4, фінансова ефективність: 3, ринкові можливості: 5, конкуренція: 2, соціальна значущість: 4. Призначаються ваги кожному критерію в залежності від його важливості (наприклад, в сумі 1): технічна реалізованість: 0.25, фінансова ефективність: 0.30, ринкові можливості: 0.25, конкуренція: 0.10, соціальна значущість: 0.10. Обчислюється загальна оцінка привабливості проекту, використовуючи формулу: Загальна оцінка = (Оцінка1×Вага1) + (Оцінка2×Вага2) + (Оцінка3×Вага3) + (Оцінка4×Вага4) + (Оцінка5×Вага5)

Загальна оцінка=(4×0.25)+(3×0.30)+(5×0.25)+(2×0.10)+(4×0.10)=1.00+0.90+1.25+0.20+0.40=3.75 Загальна оцінка привабливості проекту становить 3.75 з максимально можливої оцінки 5. Це свідчить про те, що проект має високу привабливість для впровадження. Застосування експертних оцінок дозволяє об'єктивно оцінити привабливість програмної реалізації системи оптимізації продуктивності та надійності мережних систем зберігання Big Data, сприяючи прийняттю обґрунтованих рішень щодо інвестицій та подальшої реалізації проекту.

7.3 Вибір методу оцінки вартості ПЗ

Для програмної реалізації системи оптимізації продуктивності та надійності мережних систем зберігання Big Data найкраще використовувати метод оцінки на основі вартості життєвого циклу в поєднанні з методом аналогій. Це дозволить отримати більш точну оцінку вартості проекту з урахуванням всіх етапів його реалізації та експлуатації, а також зрозуміти, які витрати можуть виникнути в процесі його впровадження.

Рисунок 7.3 – Методи оцінки вартості

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Приклад економічної ефективності від впровадження системи оптимізації продуктивності та надійності мережних систем зберігання Big Data може включати аналіз витрат та вигод, що дозволяє оцінити, наскільки вигідно інвестувати в дану технологію. На рисунку 7.4. зробимо розгорнутий перелік ключових економічних вигод, які можуть бути отримані в результаті впровадження системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

У даному прикладі, впровадження системи оптимізації продуктивності та надійності мережних систем зберігання Big Data дозволило досягти значних економічних вигод, що відображається в показниках ROI і NPV. За замовченням

зазначені економічні вигоди гарантуватимуть високий ROI (250%), що вказуватиме на високу рентабельність інвестицій, що робить проект привабливим з економічної точки зору. Це свідчить про те, що інвестиції в оптимізацію продуктивності та надійності мережних систем зберігання Big Data можуть бути виправданими і вигідними для компанії.

Рисунок 7.4 – Економічна ефективність від реалізації проєкту для клієнта

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Покроковий алгоритм просування проекту програмної реалізації системи управління подано на рисунку 7.5. Цей алгоритм просування проекту програмної реалізації системи оптимізації продуктивності та надійності мережних систем зберігання Big Data забезпечує систематичний підхід до розробки, впровадження та просування рішення. Виконуючи ці етапи, можна забезпечити успіх проекту, задовольнити потреби клієнтів та досягти поставлених бізнес-цілей.

1. Аналіз ринку та потреб

- **Дослідження ринку**
- **Визначення цільової аудиторії**
- **Оцінка потреб**

2. Формування команди проекту

- **Вибір експертів**
- **Розподіл ролей**

3. Розробка концепції проекту

- **Створення технічного завдання (ТЗ)**
- **Визначення архітектури системи**

4. Планування та бюджетування

- **Оцінка витрат**
- **Складання плану реалізації**

5. Розробка маркетингової стратегії

- **Брендинг проекту**
- **Розробка комунікаційної стратегії**

6. Впровадження проекту

- **Розробка ПЗ**
- **Впровадження на об'єктах**
- **Навчання персоналу**

7. Промоція та продаж

- **Запуск рекламної кампанії**
- **Проведення демонстрацій**
- **Участь у виставках та конференціях**

8. Моніторинг та підтримка

- **Збір відгуків**
- **Оцінка результатів**
- **Постійна підтримка та оновлення**

9. Аналіз та корекція стратегії

- **Оцінка ефективності маркетингових кампаній**
- **Адаптація продукту**

Рисунок 7.5 – Алгоритм просування проєкту

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Оптимізація каналів збуту та шляхів реалізації проекту програмної реалізації системи оптимізації продуктивності та надійності мережних систем

зберігання Big Data може включати кілька стратегій, спрямованих на підвищення ефективності, зниження витрат та покращення взаємодії з клієнтами:

– аналіз та сегментація цільового ринку;
– використання цифрових каналів збуту;
– партнерство та альянси;
– офлайн-канали збуту;
– покращення обслуговування клієнтів;
– автоматизація процесів;
– гнучкі моделі ціноутворення;
– зворотний зв’язок та вдосконалення.

Оптимізація каналів збуту для системи оптимізації продуктивності та надійності мережних систем зберігання Big Data передбачає комплексний підхід, що охоплює як онлайн, так і офлайн стратегії. Використання технологій, партнерських відносин та поліпшення обслуговування клієнтів можуть суттєво підвищити ефективність реалізації проекту і допомогти досягти поставлених бізнес-цілей.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключові фактори успіху проекту програмної реалізації системи оптимізації продуктивності та надійності мережних систем зберігання Big Data (гиперконвергентного дата-центра) мають включати пункти запропоновані на рисунку 7.6.

Успіх проекту програмної реалізації системи оптимізації продуктивності та надійності мережних систем зберігання Big Data залежить від інтеграції всіх цих факторів. Чітке планування, ефективна команда, технологічні інновації та підтримка з боку керівництва – все це є критично важливим для досягнення позитивного результату.

Рисунок 7.6 – Ключові фактори успіху проекту

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Електронно-обчислювальна машина (ЕОМ) відіграє важливу роль у житті сучасної людини. Кожного дня мільйони людей використовують ЕОМ для пошуку необхідної інформації, спілкуванні у соціальних мережах, перегляду новин, роботи тощо. Багато людей користуються ЕОМ у професійних цілях, оскільки завдяки ЕОМ з’явилося багато нових професій.

Аналізуючи умови працівників ІТ-сфери, на перший погляд, може здатися, що працівники сфери інформаційних технологій не схильні до ризиків на виробництві, та якщо більш глибоко розглянути умови і специфіку праці фахівців сфері ІТ-індустрії, можна виявити ряд факторів які будуть мати негативний сплив на стан охорони праці, та на самого ІТ-фахівця зокрема. Сюди можна віднести як невідповідність освітлення, так і високий рівень шуму, що негативно позначатимуться як на емоційному так і на фізичному стані фахівця, призводитимуть до зниження ефективності праці та виробничих травм. Також, важливим моментом охорони праці ІТ-фахівця є врахування його психологічних можливостей (швидкість реакції, особливості пам'яті та уваги, емоційний стан, тощо). Для того, щоб забезпечити ефективну роботу ІТ-фахівця, потрібно враховувати та максимально компенсувати такі негативні фактори як: надмірне нервово-емоційне навантаження, довготривалі статичні перевантаження, обмежена рухова активність. Всі ці чинники призводить до різноманітних відхилень у стані здоров’я, зокрема до перевтоми, зниження фізичної та розумової працездатності, неврозів, захворювань серцево-судинної системи тощо. Метою даного розділу є огляд конкретних умов праці спеціаліста у сфері ІТ-індустрії. Завданнями для даного розділу є: аналіз умов праці на робочому місці фахівця ІТ-індустрії, розробка конкретних рекомендацій щодо покращення умов

праці фахівців ІТ-індустрії, огляд пожежної безпеки на ІТ-підприємстві та розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ-фахівці.

8.2 Аналіз умов праці на робочому місці ІТ-фахівця

На робочому місці ІТ-фахівця (або програміста) виникають небезпечні та шкідливі для безпечної життєдіяльності фактори:

– підвищений рівень шуму;

– несприятливі мікрокліматичні умови;

– недостатній рівень освітленості;

– шкідливі речовини;

– підвищений рівень електромагнітних випромінювань радіочастот;

– висока напруга електричної мережі;

– статична електрика та інші.

Робота програміста супроводжується також підвищеним ступенем напруженості трудового процесу. При систематичному впливі виробничих факторів, які не відповідають нормативним показникам, зростає рівень професійно зумовленої захворюваності працюючих та можуть виникнути професійні захворювання органів зору, руху, нервової системи. Таким чином, вивчення умов праці на робочому місці програміста є необхідною умовою запобігання негативних наслідків впливу небезпечних та шкідливих факторів. Робоче місце, добре пристосоване до трудової діяльності інженера, правильно і доцільно організоване, щодо простору, форми, розміру забезпечує йому зручне положення при роботі і високу продуктивність праці при найменшому фізичному і психічному напруженні.

Нормування параметрів проводиться в залежності від періоду року та категорії важкості виконуваних робіт. Для постійних робочих місць, якими є робочі місця ІТ-фахівців, встановлені оптимальні параметри мікроклімату, а за

неможливості їх дотримання використовують допустимі параметри. Робота ІТ-фахівця за важкістю відноситься до Іа (роботи, що виконуються сидячи і не потребують фізичного напруження) та Іб (роботи, що виконуються сидячи, стоячи або пов'язані з ходінням та супроводжуються деяким фізичним напруженням) категорій. В таблиці 8.1. наведені оптимальні параметри мікроклімату в приміщеннях.

Таблиця 8.1 – Параметри мікроклімату для приміщень з ПК

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	22...24°С; 40... 60%; до 0,1 м/с
Теплий	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	23...25 °С; 40...60%; 0,1...0,2 м/с

Виміряні за допомогою приладів температура та вологість у приміщеннях праці ІТ-фахівців повинні відповідати зазначеним у таблиці для теплого періоду року. Для нормалізації параметрів мікроклімату слід використовувати у приміщеннях кондиціювання повітря, або забезпечити подачу свіжого повітря системами вентиляції. Норми подачі свіжого повітря наведені у таблиці 8.2.

Таблиця 8.2 – Норми подачі свіжого повітря в приміщення

Характеристика приміщення	Об'ємна витрата свіжого повітря, що подається в приміщення, $м^3$, на одну людину в годину
Об'єм до 20 $м^3$ на людину 20... 40 $м^3$ на людину Більше 40 $м^3$на людину	Не менше 30 Не менше 20 Може біти використана природна вентиляція

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення їх привабливості, позитивно впливає на продуктивність праці. Забарвлення приміщень і меблів повинні створювати сприятливі умови для зорового сприйняття, гарного настрою. У службових приміщеннях, у яких виконується одноманітна розумова робота, що вимагає значної нервової напруги і великого зосередження, забарвлення повинно бути спокійних тонів – малонасичені відтінки холодного зеленого або блакитного кольорів.

При розробці оптимальних умов праці програміста необхідно враховувати освітленість. Раціональне освітлення робочого місця є одним з найважливіших факторів, що впливають на ефективність трудової діяльності людини, що попереджають травматизм і професійні захворювання. Правильно організоване освітлення створює сприятливі умови праці, підвищує працездатність і продуктивність праці. Освітлення на робочому місці програміста повинно бути таким, щоб працівник міг без напруги зору виконувати свою роботу. Стомлюваність органів зору залежить від ряду причин: недостатність освітленості; надмірна освітленість; неправильний напрям світла. Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або професійних захворювань. [2]

8.3 Пропозиції щодо підвищення працездатності ІТ-фахівців

Поява та впровадження нових інформаційно-комунікаційних технологій зумовлює необхідність подальшого вдосконалення охорони праці фахівців ІТ-індустрії. Все це потребує розробки нових нормативно-правових актів з регламентації праці та відпочинку фахівців ІТ-індустрії і стандартів під-

приємств, центрів комп'ютерної техніки, центрів інформаційних технологій, сучасних комп'ютерних класів.

Для підвищення розумової працездатності то зорової роботи повинна здійснюватися ергономічна оптимізація в рамках системи «оператор-термінал», яка сприятиме результативній фізичній та інтелектуальний працездатності і відновленню психосоматичного здоров'я фахівців ІТ-індустрії. Всі наведені заходи щодо вдосконалення охорони праці фахівців ІТ-індустрії повинні контролюватися службою охорони праці та комісією з охорони праці підприємства. Особливе значення у соціальному захисті цієї категорії працівників належить прийняття комплексного договору, який може забезпечити фахівців додатковими пільгами та компенсаціями.

Для більшого розуміння, пропозиції щодо підвищення працездатності ІТ-фахівців, розіб'ємо на декілька категорій:

1 Середовище і розпорядок праці. Для мінімізації негативних ефектів, що пов'язані з перевтомленням ІТ-фахівців, потрібно чітко прописати і реалізувати графік періодів праці-відпочинку, щоб фахівець міг можливість переключити увагу, дати можливість відпочити очам, мозку, елементарно, встати розім'яти ноги. Також потрібно зробити максимально комфортними умови мікроклімату у офісному приміщенні, де працюють ІТ-фахівці. Мається на увазі встановлення і експлуатація, коли виникає необхідність, кондиціонерів, опалення, та системи вентиляції, задля попередження перегрівання, переохолодження ІТ-фахівців, і подальшої неможливості ними виконувати свої функції. Також, за можливості, нами пропонується введення практики віддаленої праці ІТ-фахівцями, якщо роботодавець не може забезпечити оптимальні і безпечні умови в офісному приміщенні, або якщо фахівця вони не влаштовують із певних причин.

2 Фізичні і психоемоційні чинники. Першим і найважливішим чинником, що впливає на працездатність ІТ-фахівців є робоче місце, і саме тому, роботодавець має забезпечити максимальний його комфорт і безпеку. Гарантією цих факторів може слугувати сертифікація меблів, що використовуються на підприємстві ІТ-галузі. Тому нами пропонується закупівля тільки меблів, які

пройшли сертифікацію на відповідність. Під психоемоційними чинниками ми розуміємо гарне самопочуття фахівців, позитивний настій, гарний психологічний клімат у колективі, тощо. Задля того, щоб психоемоційні чинники мали максимально позитивний ефект, керівництву слід поводити заходи, які сприятимуть укріпленню і покращенню міжособистісних стосунків у колективі, таких як психологічні тренінги, тімбілдінг, спортивні змагання і естафети. Також, сюди можна віднести розробку і впровадження системи мотивації працівників, як фінансової, так моральної і адміністративної.

8.4 Розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ-фахівці

Приміщення з ПК повинні мати природне і штучне освітлення, яке відповідало б вимогам ДБН В.2.5-28-2006 «Природне і штучне освітлення» [1], ДСАНПІН 3.3.2.007-98 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2].

Віконні прорізи повинні бути орієнтовані на північ або на північний схід, забезпечувати коефіцієнт природної освітленості (К.П.О.) не менш 1,5% і мати жалюзі або штори. Віконні прорізи повинні мати регульовані пристрої для відкривання, а також жалюзі, завіски, зовнішні козирки тощо. Приміщення із ПЕОМ повинні бути обладнані системою загального рівномірного освітлення. У виробничих і адміністративно-суспільних приміщеннях, де переважно ведеться робота з документами, допускається комбінована система штучного освітлення.

Штучне освітлення має здійснюватися системою загального рівномірного освітлення, яка включає суцільні або такі, що перериваються лінії світильників, розташованих збоку робочих місць (переважно ліворуч), паралельно лінії зору користувачів ПК. Світильники повинні мати розсіювачі світла. У світильниках місцевого освітлення можна використовувати лампи накалювання. При розміщенні ПК по периметру приміщення лінії світильників штучного освітлення

повинні розміщуватися локально над робочими місцями. Система освітлення робочого місця користувача ПК має відповідати наступним вимогам (рис. 8.1).

Освітленість на робочому столі користувача в зоні розташування документів має бути в межах 300-500 лк. Якщо цей рівень освітленості неможливо забезпечити системою загального освітлення то допускається застосування світильників місцевого освітлення, але при цьому не повинно бути відблисків на поверхні екрану (яскравість відблисків не повинна перевищувати 40 кд/м^2) та перевищення його освітленості більше ніж 300 лк.

Рисунок 8.1 – Вимоги до системи освітлення робочого місця користувача ПК

Яскравість світильників загального освітлення, а також яскравість стелі при застосуванні системи відбитого освітлення не повинна перевищувати 200 кд/м^2. Величина коефіцієнта пульсації освітленості не повинна перевищувати 5%.

Що стосується розподілу яскравості в полі зору працюючих за дисплеями ПК, то відношення значень яскравості робочих поверхонь не повинно перевищувати 3:1.

Проведемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 6 м, довжина – 7 м, висота – 2,9 м. У зазначеному приміщенні працює 7 людей.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою [1]:

$$F = E \cdot S \cdot K \cdot Z / n,$$

де F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; *E = 300 Лк;*

S – площа освітлюваного приміщення (у нашому випадку S=6 х 6,8 = 40,8 $м^2$);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K = 1,5);*

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку Z = *1,1*);

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці [7]; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють:

$$\rho_{стін} = 50\% \text{ і } \rho_{стелі} = 50\%.$$

Обчислимо індекс приміщення за формулою:

$$i = S/(h \cdot (A+B)),$$

де S – площа приміщення, S = 40,8 $м^2$*;*

h – розрахункова висота підвісу, h = 3 *м (співпадає з в*исотою стелі, т.я. лампи освітлення закріплюються на стелі)*;*

А – ширина приміщення, *А = 6 м;*

В – довжина приміщення, *В* = 6,8 *м.*

Підставимо всі значення у формулу та визначимо індекс приміщення: i=1,1.

Знаючи індекс приміщення, за знаходимо *n = 0,46 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним типом лампам) [7].* Підставимо всі значення у формулу, визначемо світловий потік: F=43904 Лм.

Для розрахунку дудемо використовувати *світлодіодні стельові панелі* Lezard 6400K 72 Вт., світловий потік яких $F_л$ = 7200 *Лм.*

Число ламп визначається по формулі:

$$N=F/F_л$$

де: F – світловий потік, $F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначемо індекса приміщення

$$N= 43904 / 7200=6,09 \text{ шт.}$$

Приймаємо необхідну кількість *світлодіодних світильників* 7 шт.

8.5 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров’я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз умов праці на робочому місці ІТ-фахівця, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров’я програміста. Розроблено заходи з умов поліпшення охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем оптимізації продуктивності та надійності мережних систем зберігання Big Data.

– Досліджена система оптимізації продуктивності та надійності мережних систем зберігання Big Data.

– На основі отриманих результатів досліджень створена програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання оптимізації продуктивності та надійності мережних систем зберігання Big Data.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

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 Visual C#. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм TEA.

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.

Проведено маркетингове та економічне обґрунтування ІТ-проєкту, що дозволило визначити ключові фактори успіху даного проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гречушкін С.В. Дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2024.

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

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

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

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

11. 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.

12. 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.

13. 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.

14. 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.

15. 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.

16. 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.

17. 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.

18. 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.

19. 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.

20. 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.

21. 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.

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

27. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. 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.

34. 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

35. 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.

36. 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.

37. 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.

38. 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.

39. 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.

40. 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.

41. 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).

42. 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

43. 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.

44. 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.

45. 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.

46. 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.

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

48. 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.

49. Смірнов О.А., Смірнов С.А. Дідик А.К., Дрєєв О.М. Моделі системи нейромережевих експертів безпечної маршрутизації у хмарних антивірусних системах. Збірник наукових праць "Системи обробки інформації". - Випуск 3 (140). - Х.: ХУПС - 2016. - С. 36-39.

50. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Спосіб контролю ліній зв'язку телекомунікаційної системи антивірусу. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. - 2016. - С. 121-127.

51. Смірнов О.А., Смірнов С.А., Дідик А.К. Метод безпечної маршрутизації метаданих у хмарні антивірусні системи. Системи озброєння та військова техніка. - Випуск 2 (46) - Х.: ХУПС - 2016. - С. 146-149.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 18-13 від 07.08.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) 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 з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи оптимізації продуктивності та надійності мережних систем зберігання Big Data;

– 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 забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 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 Visual C#.

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 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати маркетингове та економічне обґрунтування ІТ-проєкту з урахуванням цін на 3 вересня 2024 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ-фахівці.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 96 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Маркетингове та економічне обґрунтування ІТ-проєкту.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 02.12.2024 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 19.12.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

_____ Коваленко А.С.

Дослідження та програмна реалізація системи оптимізації продуктивності та надійності мережних систем зберігання Big Data

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 56

Літера: РП

Кропивницький – 2024 року

Файл Code_Rid_Solomon_Encoder.cs - кодер коду Ріда-Соломона для системи оптимізації продуктивності та надійності мережних систем зберігання Big Data

```
using System;
using System.Threading;

namespace Data_Center
{
    /// <summary>
    /// Клас кодера коду Ріда-Соломона
    /// </summary>
    public class Code_Rid_Solomon_Encoder : Code_Rid_Solomon_Base
    {
        #region Construction & Destruction

        /// <summary>
        /// Конструктор кодера за замовчуванням
        /// </summary>
        public Code_Rid_Solomon_Encoder()
        {
            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        /// <summary>
        /// Базовий конструктор кодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        public Code_Rid_Solomon_Encoder(int dataCount, int eccCount)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount,
(int)Code_Rid_Solomon_Type.Alternative);

            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        /// <summary>
        /// Розширений конструктор кодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="codecType">Тип кодера коду Ріда-Соломона (по типу
матриці)</param>
        public Code_Rid_Solomon_Encoder(int dataCount, int eccCount, int
codecType)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, codecType);

            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Установка конфігурації кодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
```

```
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="codecType">Тип кодека кодера коду Ріда-Соломона (по
типу матриці)</param>
        /// <returns>Булевський прапор операції установки конфігурації</returns>
        public bool SetConfig(int dataCount, int eccCount, int codecType)
        {
            int maxVolCount;

            // Установлюємо константи, що відповідають обраному режиму
            if (codecType == (int)Code_Rid_Solomon_Type.Dispersal)
            {
                maxVolCount = (int)Code_Rid_Solomon_Const.MaxVolCountDisp;

            } else
            {
                maxVolCount = (int)Code_Rid_Solomon_Const.MaxVolCountAlt;
            }

            // Перевіряємо конфігурацію на коректність
            if (
                    (dataCount > 0)
                 &&
                    (eccCount > 0)
                 &&
                    ((dataCount + eccCount) <= maxVolCount)
                )
            {
                // Якщо основна конфігурація змінилася - сповіщаємо про це
                if (
                        (dataCount != this.n)
                     ||
                        (eccCount != this.m)
                     ||
                        (codecType != this.eCode_Rid_Solomon_Type)
                    )
                {
                    this.mainConfigChanged = true;
                }

                // Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eCode_Rid_Solomon_Type = codecType;

                // Також перераховуємо кількість ітерацій всіх стадій підготовки
                double n = this.n;
                double m = this.m;

                // Нормалізуємо значення для розрахунку, щоб уникнути
переповнення змінних
                NormalizeNM(ref n, ref m);

                // Кількість ітерацій на першій стадії залежить від типу
використовуваної матриці
                if (this.eCode_Rid_Solomon_Type ==
(int)Code_Rid_Solomon_Type.Alternative)
                {
                    this.iterOfFirstStage = m;

                } else
                {
                    this.iterOfFirstStage = ((n * m) * n) + (n * ((n + m) + (n *
(n + m))));
                }

                this.iterOfSecondStage = 0; // У кодери немає інвертування
матриці

                this.configIsOK = true;
```

```
            } else
            {
                //...указуємо на помилку конфігурації
                this.configIsOK = false;
            }

            return this.configIsOK;
        }

        /// <summary>
        /// Метод множення матриці кодування на вхідний прологарифмований вектор
        /// </summary>
        /// <param name="dataLog">Прологарифмований вхідний вектор (вихідні
дані)</param>
        /// <param name="ecc">Вихідний вектор (надлишкові дані)</param>
        /// <returns>Булевський прапор результату операції</returns>
        public bool Process(int[] dataLog, ref int[] ecc)
        {
            // Якщо кодер зконфігуровано некоректно, обробка неможлива!
            if (!this.configIsOK)
            {
                return false;
            }

            // Копіюємо покажчик на масив експонент для скорочення часу обігу
            int[] GF16Exp = this.eGF16.GFExpTable;

            // Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.m; i++)
            {
                int mulSum = 0;         // Сума добутку рядка матриці на
стовпець
                int i_n = i * this.n;   // Зсув у масиві до елементів i-ой рядка

                for (int j = 0; j < this.n; j++)
                {
                    mulSum ^= GF16Exp[this.FLog[i_n + j] + dataLog[j]];
                }

                ecc[i] = mulSum;
            }

            return true;
        }

        #endregion Public Operations

        #region Private Operations

        /// <summary>
        /// Заповнення матриці Вандермонда даними
        /// </summary>
        protected override void FillFLog()
        {
            // Якщо основна конфігурація змінилася...
            if (this.mainConfigChanged)
            {
                if (this.eCode_Rid_Solomon_Type ==
(int)Code_Rid_Solomon_Type.Dispersal)
                {
                    //...робимо формування дисперсної матриці "D"
                    if (!MakeDispersalMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
```

```
                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }

                } else
                {
                    //...робимо формування альтернативного заповнення матриці
"A"
                    if (!MakeAlternativeMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                // Виділяємо пам'ять під матрицю "FLog"
                this.FLog = new int[this.m * this.n];

                // Заповнюємо матрицю кодування
                for (int i = 0; i < this.m; i++)
                {
                    // Зсув у масиві до елементів i-ой рядка
                    int i_n = i * this.n;

                    // Залежно від типу кодера беремо дані з відповідного масиву
                    if (this.eCode_Rid_Solomon_Type ==
(int)Code_Rid_Solomon_Type.Dispersal)
                    {
                        // Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
                            // У матрицю кодування поміщаємо логарифми її
вихідних елементів
                            // (для прискорення множення матриці на вектор)
                            this.FLog[i_n + j] = this.eGF16.Log(this.D[((this.n
+ i) * this.n) + j]);
                        }

                    } else
                    {
                        // Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
                            int idx = i_n + j;

                            // У матрицю кодування поміщаємо логарифми її
вихідних елементів
                            // (для прискорення множення матриці на вектор)
                            this.FLog[idx] = this.eGF16.Log(this.A[idx]);
                        }
                    }

                    // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                    // буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
```

```
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                // Якщо є передплата на делегата завершення...
                if (OnCode_Rid_Solomon_MatrixFormingFinish != null)
                {
                    //...повідомляємо, що екземпляр класу готовий до роботи
                    OnCode_Rid_Solomon_MatrixFormingFinish();
                }

                //...і скидаємо прапор
                this.mainConfigChanged = false;
            }

            // Якщо є передплата на делегата завершення...
            if (OnCode_Rid_Solomon_MatrixFormingFinish != null)
            {
                //...повідомляємо, що екземпляр класу готовий до роботи
                OnCode_Rid_Solomon_MatrixFormingFinish();
            }

            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;

            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }

        #endregion Private Operations
    }
}
```

Файл ProcessForm.cs - вікно відображення процесів запису/читання та перевірки цілісності даних у мережних системах зберігання

```
namespace Data_Center
{
    partial class ProcessForm
    {
        /// <summary>
                /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true якщо керуючі ресурси повинні бути
розташовані, у іншому випадку  false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Необхідний метод для підтримки розробника - не модифікується
        /// зміст цього метода використовується редактором коду.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(ProcessForm));
            this.processPriorityGroupBox = new System.Windows.Forms.GroupBox();
            this.processPriorityComboBox = new System.Windows.Forms.ComboBox();
            this.processGroupBox = new System.Windows.Forms.GroupBox();
            this.processProgressBar = new System.Windows.Forms.ProgressBar();
            this.fileAnalyzeStatGroupBox = new System.Windows.Forms.GroupBox();
            this.percOfAltEccLabel = new System.Windows.Forms.Label();
            this.percOfDamageLabel = new System.Windows.Forms.Label();
            this.percOfAltEccLabel_ = new System.Windows.Forms.Label();
            this.percOfDamageLabel_ = new System.Windows.Forms.Label();
            this.logGroupBox = new System.Windows.Forms.GroupBox();
            this.logListBox = new System.Windows.Forms.ListBox();
            this.countGroupBox = new System.Windows.Forms.GroupBox();
            this.errorCountLabel = new System.Windows.Forms.Label();
            this.okCountLabel = new System.Windows.Forms.Label();
            this.errorPictureBox = new System.Windows.Forms.PictureBox();
            this.okPictureBox = new System.Windows.Forms.PictureBox();
            this.errorCountLabel_ = new System.Windows.Forms.Label();
            this.okCountLabel_ = new System.Windows.Forms.Label();
            this.toolTip = new System.Windows.Forms.ToolTip(this.components);
            this.stopButton_Windows_8 = new PinkieControls.Button_Windows_8();
            this.pauseButton_Windows_8 = new PinkieControls.Button_Windows_8();
            this.closingTimer = new System.Windows.Forms.Timer(this.components);
            this.processTimer = new System.Windows.Forms.Timer(this.components);
            this.processPriorityGroupBox.SuspendLayout();
            this.processGroupBox.SuspendLayout();
            this.fileAnalyzeStatGroupBox.SuspendLayout();
            this.logGroupBox.SuspendLayout();
            this.countGroupBox.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.errorPictureBox)).BeginInit();
```

```
((System.ComponentModel.ISupportInitialize)(this.okPictureBox)).BeginInit();
            this.SuspendLayout();
            //
            // processPriorityGroupBox
            //

this.processPriorityGroupBox.Controls.Add(this.processPriorityComboBox);
            this.processPriorityGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.processPriorityGroupBox.Location = new
System.Drawing.Point(617, 216);
            this.processPriorityGroupBox.Name = "processPriorityGroupBox";
            this.processPriorityGroupBox.Size = new System.Drawing.Size(135,
64);
            this.processPriorityGroupBox.TabIndex = 0;
            this.processPriorityGroupBox.TabStop = false;
            this.processPriorityGroupBox.Text = "Пріоритет процесу";
            //
            // processPriorityComboBox
            //
            this.processPriorityComboBox.BackColor =
System.Drawing.SystemColors.Control;
            this.processPriorityComboBox.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
            this.processPriorityComboBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.processPriorityComboBox.FormattingEnabled = true;
            this.processPriorityComboBox.Items.AddRange(new object[] {
            "За замовчуванням",
            "Знижений",
            "Нормальний",
            "Підвищений",
            "Найвищий"});
            this.processPriorityComboBox.Location = new System.Drawing.Point(9,
33);
            this.processPriorityComboBox.Name = "processPriorityComboBox";
            this.processPriorityComboBox.Size = new System.Drawing.Size(117,
21);
            this.processPriorityComboBox.TabIndex = 0;
            this.processPriorityComboBox.TabStop = false;
            this.toolTip.SetToolTip(this.processPriorityComboBox, "Список
можливих значень пріоритету процесу обробки");
            this.processPriorityComboBox.SelectedIndexChanged += new
System.EventHandler(this.processPriorityComboBox_SelectedIndexChanged);
            //
            // processGroupBox
            //
            this.processGroupBox.Controls.Add(this.processProgressBar);
            this.processGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.processGroupBox.Location = new System.Drawing.Point(12, 9);
            this.processGroupBox.Name = "processGroupBox";
            this.processGroupBox.Size = new System.Drawing.Size(871, 65);
            this.processGroupBox.TabIndex = 0;
            this.processGroupBox.TabStop = false;
            this.processGroupBox.Text = "Обробка";
            //
            // processProgressBar
            //
            this.processProgressBar.Location = new System.Drawing.Point(14, 30);
            this.processProgressBar.Name = "processProgressBar";
            this.processProgressBar.Size = new System.Drawing.Size(844, 20);
            this.processProgressBar.Style =
System.Windows.Forms.ProgressBarStyle.Continuous;
            this.processProgressBar.TabIndex = 0;
            //
            // fileAnalyzeStatGroupBox
            //
```

```
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfAltEccLabel);
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfDamageLabel);
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfAltEccLabel_);
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfDamageLabel_);
            this.fileAnalyzeStatGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.fileAnalyzeStatGroupBox.Location = new System.Drawing.Point(12,
216);
            this.fileAnalyzeStatGroupBox.Name = "fileAnalyzeStatGroupBox";
            this.fileAnalyzeStatGroupBox.Size = new System.Drawing.Size(459,
64);
            this.fileAnalyzeStatGroupBox.TabIndex = 0;
            this.fileAnalyzeStatGroupBox.TabStop = false;
            this.fileAnalyzeStatGroupBox.Text = "Результат аналізу цілісності
даних";
            //
            // percOfAltEccLabel
            //
            this.percOfAltEccLabel.AutoSize = true;
            this.percOfAltEccLabel.Location = new System.Drawing.Point(195, 41);
            this.percOfAltEccLabel.Name = "percOfAltEccLabel";
            this.percOfAltEccLabel.Size = new System.Drawing.Size(10, 13);
            this.percOfAltEccLabel.TabIndex = 0;
            this.percOfAltEccLabel.Text = "-";
            //
            // percOfDamageLabel
            //
            this.percOfDamageLabel.AutoSize = true;
            this.percOfDamageLabel.Location = new System.Drawing.Point(195, 20);
            this.percOfDamageLabel.Name = "percOfDamageLabel";
            this.percOfDamageLabel.Size = new System.Drawing.Size(10, 13);
            this.percOfDamageLabel.TabIndex = 0;
            this.percOfDamageLabel.Text = "-";
            //
            // percOfAltEccLabel_
            //
            this.percOfAltEccLabel_.AutoSize = true;
            this.percOfAltEccLabel_.Location = new System.Drawing.Point(7, 41);
            this.percOfAltEccLabel_.Name = "percOfAltEccLabel_";
            this.percOfAltEccLabel_.Size = new System.Drawing.Size(163, 13);
            this.percOfAltEccLabel_.TabIndex = 0;
            this.percOfAltEccLabel_.Text = "Резерв перевірочних даних для
відновлення:";
            //
            // percOfDamageLabel_
            //
            this.percOfDamageLabel_.AutoSize = true;
            this.percOfDamageLabel_.Location = new System.Drawing.Point(7, 20);
            this.percOfDamageLabel_.Name = "percOfDamageLabel_";
            this.percOfDamageLabel_.Size = new System.Drawing.Size(148, 13);
            this.percOfDamageLabel_.TabIndex = 0;
            this.percOfDamageLabel_.Text = "Всього пошкоджених секторів:";
            //
            // logGroupBox
            //
            this.logGroupBox.Controls.Add(this.logListBox);
            this.logGroupBox.Location = new System.Drawing.Point(12, 80);
            this.logGroupBox.Name = "logGroupBox";
            this.logGroupBox.Size = new System.Drawing.Size(871, 130);
            this.logGroupBox.TabIndex = 0;
            this.logGroupBox.TabStop = false;
            this.logGroupBox.Text = "Лог процесу";
            //
            // logListBox
            //
            this.logListBox.BackColor = System.Drawing.SystemColors.Control;
            this.logListBox.BorderStyle = System.Windows.Forms.BorderStyle.None;
            this.logListBox.FormattingEnabled = true;
            this.logListBox.HorizontalScrollbar = true;
```

```
            this.logListBox.Location = new System.Drawing.Point(7, 23);
            this.logListBox.Name = "logListBox";
            this.logListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.logListBox.Size = new System.Drawing.Size(851, 91);
            this.logListBox.TabIndex = 0;
            this.logListBox.TabStop = false;
            this.logListBox.UseTabStops = false;
            this.logListBox.SelectedIndexChanged += new
System.EventHandler(this.logListBox_SelectedIndexChanged);
            //
            // countGroupBox
            //
            this.countGroupBox.Controls.Add(this.errorCountLabel);
            this.countGroupBox.Controls.Add(this.okCountLabel);
            this.countGroupBox.Controls.Add(this.errorPictureBox);
            this.countGroupBox.Controls.Add(this.okPictureBox);
            this.countGroupBox.Controls.Add(this.errorCountLabel_);
            this.countGroupBox.Controls.Add(this.okCountLabel_);
            this.countGroupBox.Location = new System.Drawing.Point(482, 216);
            this.countGroupBox.Name = "countGroupBox";
            this.countGroupBox.Size = new System.Drawing.Size(124, 64);
            this.countGroupBox.TabIndex = 0;
            this.countGroupBox.TabStop = false;
            this.countGroupBox.Text = "Лічільник процесу";
            //
            // errorCountLabel
            //
            this.errorCountLabel.AutoSize = true;
            this.errorCountLabel.Location = new System.Drawing.Point(63, 41);
            this.errorCountLabel.Name = "errorCountLabel";
            this.errorCountLabel.Size = new System.Drawing.Size(13, 13);
            this.errorCountLabel.TabIndex = 0;
            this.errorCountLabel.Text = "0";
            this.toolTip.SetToolTip(this.errorCountLabel, "Лічильник некоректно
оброблених файлів");
            //
            // okCountLabel
            //
            this.okCountLabel.AutoSize = true;
            this.okCountLabel.Location = new System.Drawing.Point(63, 20);
            this.okCountLabel.Name = "okCountLabel";
            this.okCountLabel.Size = new System.Drawing.Size(13, 13);
            this.okCountLabel.TabIndex = 0;
            this.okCountLabel.Text = "0";
            this.toolTip.SetToolTip(this.okCountLabel, "Лічильник коректно
оброблених файлів");
            //
            // errorPictureBox
            //
            this.errorPictureBox.Image =
((System.Drawing.Image)(resources.GetObject("errorPictureBox.Image")));
            this.errorPictureBox.Location = new System.Drawing.Point(10, 40);
            this.errorPictureBox.Name = "errorPictureBox";
            this.errorPictureBox.Size = new System.Drawing.Size(21, 15);
            this.errorPictureBox.TabIndex = 2;
            this.errorPictureBox.TabStop = false;
            //
            // okPictureBox
            //
            this.okPictureBox.Image =
((System.Drawing.Image)(resources.GetObject("okPictureBox.Image")));
            this.okPictureBox.Location = new System.Drawing.Point(10, 19);
            this.okPictureBox.Name = "okPictureBox";
            this.okPictureBox.Size = new System.Drawing.Size(21, 15);
            this.okPictureBox.TabIndex = 1;
            this.okPictureBox.TabStop = false;
            //
            // errorCountLabel_
```

```
            //
            this.errorCountLabel_.AutoSize = true;
            this.errorCountLabel_.Location = new System.Drawing.Point(28, 41);
            this.errorCountLabel_.Name = "errorCountLabel_";
            this.errorCountLabel_.Size = new System.Drawing.Size(35, 13);
            this.errorCountLabel_.TabIndex = 0;
            this.errorCountLabel_.Text = "Error :";
            this.toolTip.SetToolTip(this.errorCountLabel_, "Лічильник некоректно
оброблених файлів");
            //
            // okCountLabel_
            //
            this.okCountLabel_.AutoSize = true;
            this.okCountLabel_.Location = new System.Drawing.Point(28, 20);
            this.okCountLabel_.Name = "okCountLabel_";
            this.okCountLabel_.Size = new System.Drawing.Size(28, 13);
            this.okCountLabel_.TabIndex = 0;
            this.okCountLabel_.Text = "OK :";
            this.toolTip.SetToolTip(this.okCountLabel_, "Лічильник коректно
оброблених файлів");
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // stopButton_Windows_8
            //
            this.stopButton_Windows_8.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.stopButton_Windows_8.DefaultScheme = true;
            this.stopButton_Windows_8.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.stopButton_Windows_8.Hint = "";
            this.stopButton_Windows_8.Location = new System.Drawing.Point(762,
257);
            this.stopButton_Windows_8.Name = "stopButton_Windows_8";
            this.stopButton_Windows_8.Scheme =
PinkieControls.Button_Windows_8.Schemes.Blue;
            this.stopButton_Windows_8.Size = new System.Drawing.Size(121, 23);
            this.stopButton_Windows_8.TabIndex = 2;
            this.stopButton_Windows_8.Text = "Перервати обробку";
            this.toolTip.SetToolTip(this.stopButton_Windows_8, "Припинення
обробки файлів із закриттям даного вікна");
            this.stopButton_Windows_8.Click += new
System.EventHandler(this.stopButton_Windows_8_Click);
            //
            // pauseButton_Windows_8
            //
            this.pauseButton_Windows_8.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.pauseButton_Windows_8.DefaultScheme = true;
            this.pauseButton_Windows_8.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.pauseButton_Windows_8.Hint = "";
            this.pauseButton_Windows_8.Location = new System.Drawing.Point(762,
220);
            this.pauseButton_Windows_8.Name = "pauseButton_Windows_8";
            this.pauseButton_Windows_8.Scheme =
PinkieControls.Button_Windows_8.Schemes.Blue;
            this.pauseButton_Windows_8.Size = new System.Drawing.Size(121, 23);
            this.pauseButton_Windows_8.TabIndex = 1;
            this.pauseButton_Windows_8.Text = "Пауза";
            this.toolTip.SetToolTip(this.pauseButton_Windows_8,
"Постановка/зняття процесу обробки з паузи");
```

```
            this.pauseButton_Windows_8.Click += new
System.EventHandler(this.pauseButton_Windows_8_Click);
            //
            // closingTimer
            //
            this.closingTimer.Tick += new
System.EventHandler(this.closingTimer_Tick);
            //
            // processTimer
            //
            this.processTimer.Interval = 500;
            this.processTimer.Tick += new
System.EventHandler(this.processTimer_Tick);
            //
            // ProcessForm
            //
            this.AutoScaleDimensions = new System.Drawing.Size(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(894, 292);
            this.ControlBox = false;
            this.Controls.Add(this.stopButton_Windows_8);
            this.Controls.Add(this.pauseButton_Windows_8);
            this.Controls.Add(this.countGroupBox);
            this.Controls.Add(this.processPriorityGroupBox);
            this.Controls.Add(this.logGroupBox);
            this.Controls.Add(this.fileAnalyzeStatGroupBox);
            this.Controls.Add(this.processGroupBox);
            this.DoubleBuffered = true;
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "ProcessForm";
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Обробка файлу";
            this.Load += new System.EventHandler(this.ProcessForm_Load);
            this.processPriorityGroupBox.ResumeLayout(false);
            this.processGroupBox.ResumeLayout(false);
            this.fileAnalyzeStatGroupBox.ResumeLayout(false);
            this.fileAnalyzeStatGroupBox.PerformLayout();
            this.logGroupBox.ResumeLayout(false);
            this.countGroupBox.ResumeLayout(false);
            this.countGroupBox.PerformLayout();

((System.ComponentModel.ISupportInitialize)(this.errorPictureBox)).EndInit();

((System.ComponentModel.ISupportInitialize)(this.okPictureBox)).EndInit();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox processPriorityGroupBox;
        private System.Windows.Forms.GroupBox processGroupBox;
        private System.Windows.Forms.ProgressBar processProgressBar;
        private System.Windows.Forms.GroupBox fileAnalyzeStatGroupBox;
        private System.Windows.Forms.Label percOfDamageLabel_;
        private System.Windows.Forms.Label percOfAltEccLabel_;
        private System.Windows.Forms.GroupBox logGroupBox;
        private System.Windows.Forms.GroupBox countGroupBox;
        private System.Windows.Forms.Label errorCountLabel_;
        private System.Windows.Forms.Label okCountLabel_;
        private System.Windows.Forms.ListBox logListBox;
        private System.Windows.Forms.ComboBox processPriorityComboBox;
```

```
        private System.Windows.Forms.PictureBox errorPictureBox;
        private System.Windows.Forms.PictureBox okPictureBox;
        private System.Windows.Forms.Label errorCountLabel;
        private System.Windows.Forms.Label okCountLabel;
        private System.Windows.Forms.ToolTip toolTip;
        private System.Windows.Forms.Timer closingTimer;
        private System.Windows.Forms.Label percOfAltEccLabel;
        private System.Windows.Forms.Label percOfDamageLabel;
        private PinkieControls.Button_Windows_8 pauseButton_Windows_8;
        private PinkieControls.Button_Windows_8 stopButton_Windows_8;
        private System.Windows.Forms.Timer processTimer;
    }
}
```

Файл FileAnalyzer.cs - контроль цілісності даних

```
using System;
using System.Threading;
using System.IO;

namespace Data_Center
{
    /// <summary>
    /// Клас контролю цілісності набору файлів-томів
    /// </summary>
    public class FileAnalyzer
    {
        #region Delegates

        /// <summary>
        /// Делегат відновлення прогресу контролю цілісності файлів
        /// </summary>
        public OnUpdateDoubleValueHandler OnUpdateFileAnalyzeProgress;

        /// <summary>
        /// Делегат завершення процесу контролю цілісності файлів
        /// </summary>
        public OnEventHandler OnFileAnalyzeFinish;

        /// <summary>
        /// Делегат одержання статистики ушкоджень багатотомного архіву
        /// </summary>
        public OnUpdateTwoDoubleValueHandler OnGetDamageStat;

        #endregion Delegates

        #region Public Properties & Data

        /// <summary>
        /// Булевська властивість "Файл обробляється?"
        /// </summary>
        public bool InProcessing
        {
            get
            {
                if (
                        (this.thrFileAnalyzer != null)
                     &&
                        (
                              (this.thrFileAnalyzer.ThreadState ==
ThreadState.Running)
                           ||
                              (this.thrFileAnalyzer.ThreadState ==
ThreadState.WaitSleepJoin)
                        )
                    )
                {
                    return true;

                } else
                {
                    return false;
                }
            }
        }

        /// <summary>
        /// Булевська властивість "Екземпляр класу закінчив обробку
        /// (має актуальний стан змінних-членів)?"
        /// </summary>
```

```
public bool Finished
{
    get
    {
        // Якщо клас не зайнятий обробкою - повертаємо значення
        if (!InProcessing)
        {
            return this.finished;

        } else
        {
            return false;
        }
    }
}

/// <summary>
/// Екземпляр класу повністю закінчив обробку?
/// </summary>
private bool finished;

/// <summary>
/// Булевська властивість "Безліч файлів оброблена коректно?"
/// </summary>
public bool ProcessedOK
{
    get
    {
        // Якщо клас не зайнятий обробкою - повертаємо значення
        if (!InProcessing)
        {
            return this.processedOK;

        } else
        {
            return false;
        }
    }
}

/// <summary>
/// Обробка набору файлів зроблена коректно?
/// </summary>
private bool processedOK;

/// <summary>
/// Список порядкових номерів наявних томів
/// </summary>
public int[] VolList
{
    get
    {
        if (!InProcessing)
        {
            return this.volList;

        } else
        {
            return null;
        }
    }
}

/// <summary>
/// Вектор, що вказує на состав томів
/// </summary>
private int[] volList;

/// <summary>
```

```
        /// Всі томи для відновлення коректні?
        /// </summary>
        public bool AllEccVolsOK
        {
            get
            {
                if (!InProcessing)
                {
                    return this.allEccVolsOK;

                } else
                {
                    return false;
                }
            }
        }

        /// <summary>
        /// Всі томи для відновлення коректні?
        /// </summary>
        private bool allEccVolsOK;

        /// <summary>
        /// Пріоритет процесу
        /// </summary>
        public int ThreadPriority
        {
            get
            {
                return (int)this.threadPriority;
            }

            set
            {
                if (
                        (this.thrFileAnalyzer != null)
                     &&
                        (this.thrFileAnalyzer.IsAlive)
                    )
                {
                    switch (value)
                    {
                        default:
                        case 0:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Lowest;

                            break;
                        }

                        case 1:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.BelowNormal;

                            break;
                        }

                        case 2:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Normal;

                            break;
                        }

                        case 3:
                        {
```

```
                            this.threadPriority =
System.Threading.ThreadPriority.AboveNormal;

                            break;
                        }

                        case 4:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Highest;

                            break;
                        }
                    }

                    // Установлюємо обраний пріоритет процесу
                    this.thrFileAnalyzer.Priority = this.threadPriority;

                    // Дублюємо установку параметра для підконтрольного об'єкта
                    if (this.eFileIntegrityCheck != null)
                    {
                        this.eFileIntegrityCheck.ThreadPriority = value;
                    }
                }
            }
        }

        /// <summary>
        /// Пріоритет процесу контролю цілісності файлів
        /// </summary>
        private ThreadPriority threadPriority;

        /// <summary>
        /// Подія, установлювана по завершенню обробки
        /// </summary>
        public ManualResetEvent[] FinishedEvent
        {
            get
            {
                return this.finishedEvent;
            }
        }

        /// <summary>
        /// Подія, установлювана по завершенню обробки
        /// </summary>
        private ManualResetEvent[] finishedEvent;

        #endregion Public Properties & Data

        #region Data

        /// <summary>
        /// Модуль для впакування (розпакування) ім'я файлу в префіксний формат
        /// </summary>
        private FileNamer eFileNamer;

        /// <summary>
        /// Екземпляр класу контролю цілісності набору файлів
        /// </summary>
        private FileIntegrityCheck eFileIntegrityCheck;

        /// <summary>
        /// Шлях до файлів для обробки
        /// </summary>
        private String path;

        /// <summary>
        /// Ім'я файлу, якому належить безліч томів
```

```
        /// </summary>
        private String fileName;

        /// <summary>
        /// Кількість основних томів
        /// </summary>
        private int dataCount;

        /// <summary>
        /// Кількість томів для відновлення
        /// </summary>
        private int eccCount;

        /// <summary>
        /// Тип кодера коду Ріда-Соломона (по типу використовуваної матриці
кодування)
        /// </summary>
        private int codecType;

        /// <summary>
        /// Використовується швидке добування з томів (без перевірки CRC-64)?
        /// </summary>
        private bool fastExtraction;

        /// <summary>
        /// Потік контролю цілісності файлу
        /// </summary>
        private Thread thrFileAnalyzer;

        /// <summary>
        /// Подія припинення обробки файлів
        /// </summary>
        private ManualResetEvent[] exitEvent;

        /// <summary>
        /// Подія продовження обробки файлів
        /// </summary>
        private ManualResetEvent[] executeEvent;

        /// <summary>
        /// Подія "пробудження" циклу очікування
        /// </summary>
        private ManualResetEvent[] wakeUpEvent;

        #endregion Data

        #region Construction & Destruction

        /// <summary>
        /// Конструктор класу перевірки цілісності набору файлів
        /// </summary>
        public FileAnalyzer()
        {
            // Модуль для впакування (розпакування) ім'я файлу в префіксний
формат
            this.eFileNamer = new FileNamer();

            // Створюємо екземпляр класу контролю цілісності набору файлів
            this.eFileIntegrityCheck = new FileIntegrityCheck();

            // Шлях до файлів для обробки за замовчуванням порожній
            this.path = "";

            // Ініціалізуємо ім'я файлу за замовчуванням
            this.fileName = "NONAME";

            // Спочатку всі томи для відновлення вважаємо ушкодженими
            this.allEccVolsOK = false;
```

```
            // Екземпляр класу повністю закінчив обробку?
            this.finished = true;

            // Обробка зроблена коректно?
            this.processedOK = false;

            // За замовчуванням встановлюється фоновий пріоритет
            this.threadPriority = 0;

            // Ініціалізуємо подію припинення обробки файлів
            this.exitEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

            // Ініціалізуємо подію продовження обробки файлів
            this.executeEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

            // Ініціалізуємо подію "пробудження" циклу очікування
            this.wakeUpEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

            // Подія, установлювана по завершенню обробки
            this.finishedEvent = new ManualResetEvent[] { new
ManualResetEvent(true) };
        }

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Метод запуску потоку обробки обчислення й записи CRC64 у кінець
файлів
        /// </summary>
        /// <param name="path">Шлях до файлів для обробки</param>
        /// <param name="fileName">Ім'я файлу для обробки</param>
        /// <param name="dataCount">Конфігурація кількості основних
томів</param>
        /// <param name="eccCount">Конфігурація кількості томів для
відновлення</param>
        /// <param name="codecType">Тип кодера коду Ріда-Соломона (по типу
матриці)</param>
        /// <param name="runAsSeparateThread">Запускати в окремому
потоці?</param>
        /// <returns>Булевський прапор операції</returns>
        public bool StartToWriteCRC64(String path, String fileName, int
dataCount, int eccCount, int codecType, bool runAsSeparateThread)
        {
            // Якщо потік обчислення CRC-64 працює - не дозволяємо повторний
запуск
            if (InProcessing)
            {
                return false;
            }

            // Скидаємо прапор коректності результату перед запуском потоку
            this.processedOK = false;

            // Скидаємо індикатор актуального стану змінних-членів
            this.finished = false;

            // Зберігаємо шлях до файлів для обробки
            if (path == null)
            {
                this.path = "";

            } else
            {
                // Робимо виділення шляху з "path" у випадку,
```

```
                // якщо туди було записано повне ім'я
                this.path = this.eFileNamer.GetPath(path);
            }

            if (fileName == null)
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return false;
            }

            // Робимо виділення короткого ім'я файлу з "fileName" у випадку,
            // якщо туди було записано повне ім'я
            this.fileName = this.eFileNamer.GetShortFileName(fileName);

            // Перевіряємо на некоректну конфігурацію
            if (
                    (dataCount <= 0)
                 ||
                    (eccCount <= 0)
                 ||
                    ((dataCount + eccCount) >
(int)Code_Rid_Solomon_Const.MaxVolCountAlt)
                )
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return false;
            }

            // Зберігаємо кількість основних томів
            this.dataCount = dataCount;

            // Зберігаємо кількість томів для відновлення
            this.eccCount = eccCount;

            // Зберігаємо тип кодера коду Ріда-Соломона (по типу
використовуваної матриці кодування)
            this.codecType = codecType;

            // Указуємо, що потік повинен виконуватися
            this.exitEvent[0].Reset();
            this.executeEvent[0].Set();
            this.wakeUpEvent[0].Reset();
            this.finishedEvent[0].Reset();

            // Якщо зазначено, що не потрібен запуск в окремому потоці,
            // запускаємо в даному
            if (!runAsSeparateThread)
            {
                // Обчислюємо CRC-64 для кожного з файлів набору
                WriteCRC64();

                // Повертаємо результат обробки
                return this.processedOK;
            }

            // Створюємо потік обчислення й запису CRC-64...
            this.thrFileAnalyzer = new Thread(new ThreadStart(WriteCRC64));

            //...потім даємо йому ім'я...
```

```
            this.thrFileAnalyzer.Name = "FileAnalyzer.WriteCRC64()";

            //...установлюємо обраний пріоритет завдання...
            this.thrFileAnalyzer.Priority = this.threadPriority;

            //...і запускаємо його
            this.thrFileAnalyzer.Start();

            // Повідомляємо, що все нормально
            return true;
        }

        /// <summary>
        /// Метод запуску потоку обробки перевірки CRC64, записаного в кінець
        /// кожного з файлів набору, з генеруванням списку наявних томів
"volList",
        /// який буде використаний декодером для відновлення даних
        /// </summary>
        /// <param name="path">Шлях до файлів для обробки</param>
        /// <param name="fileName">Ім'я файлу для обробки</param>
        /// <param name="dataCount">Конфігурація кількості основних
томів</param>
        /// <param name="eccCount">Конфігурація кількості томів для
відновлення</param>
        /// <param name="codecType">Тип кодера коду Ріда-Соломона (по типу
матриці)</param>
        /// <param name="fastExtraction">Використовується швидке добування з
томів (без перевірки CRC-64)?</param>
        /// <param name="runAsSeparateThread">Запускати в окремому
потоці?</param>
        /// <returns>Булевський прапор операції</returns>
        public bool StartToAnalyzeCRC64(String path, String fileName, int
dataCount, int eccCount, int codecType, bool fastExtraction, bool
runAsSeparateThread)
        {
            // Якщо потік обчислення CRC-64 працює - не дозволяємо повторний
запуск
            if (InProcessing)
            {
                return false;
            }

            // Спочатку всі томи для відновлення вважаємо ушкодженими
            this.allEccVolsOK = false;

            // Скидаємо прапор коректності результату перед запуском потоку
            this.processedOK = false;

            // Скидаємо індикатор актуального стану змінних-членів
            this.finished = false;

            // Зберігаємо шлях до файлів для обробки
            if (path == null)
            {
                this.path = "";

            } else
            {
                // Робимо виділення шляху з "path" у випадку,
                // якщо туди було записано повне ім'я
                this.path = this.eFileNamer.GetPath(path);
            }

            if (fileName == null)
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
```

```
                this.finishedEvent[0].Set();

                return false;
            }

            // Робимо виділення короткого ім'я файлу з "fileName" у випадку,
            // якщо туди було записано повне ім'я
            this.fileName = this.eFileNamer.GetShortFileName(fileName);

            // Перевіряємо на некоректну конфігурацію
            if (
                    (dataCount <= 0)
                 ||
                    (eccCount <= 0)
                 ||
                    ((dataCount + eccCount) >
(int)Code_Rid_Solomon_Const.MaxVolCountAlt)
                )
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return false;
            }

            // Зберігаємо кількість основних томів
            this.dataCount = dataCount;

            // Зберігаємо кількість томів для відновлення
            this.eccCount = eccCount;

            // Зберігаємо тип кодека кодера коду Ріда-Соломона (по типу
використовуваної матриці кодування)
            this.codecType = codecType;

            // Використовується швидке добування з томів (без перевірки CRC-64)?
            this.fastExtraction = fastExtraction;

            // Указуємо, що потік повинен виконуватися
            this.exitEvent[0].Reset();
            this.executeEvent[0].Set();
            this.wakeUpEvent[0].Reset();
            this.finishedEvent[0].Reset();

            // Якщо зазначено, що не потрібен запуск в окремому потоці,
            // запускаємо в даному
            if (!runAsSeparateThread)
            {
                // Обчислюємо й перевіряємо CRC-64 для кожного з файлів набору
із заповненням
                // властивості VolList
                AnalyzeCRC64();

                // Повертаємо результат обробки
                return this.processedOK;
            }

            // Створюємо потік обчислення й перевірки CRC-64...
            this.thrFileAnalyzer = new Thread(new ThreadStart(AnalyzeCRC64));

            //...потім даємо йому ім'я...
            this.thrFileAnalyzer.Name = "FileAnalyzer.AnalyzeCRC64()";

            //...установлюємо обраний пріоритет завдання...
            this.thrFileAnalyzer.Priority = this.threadPriority;
```

```
            //...і запускаємо його
            this.thrFileAnalyzer.Start();

            // Повідомляємо, що все нормально
            return true;
        }

        /// <summary>
        /// Метод зупинки потоку
        /// </summary>
        public void Stop()
        {
            // Указуємо, що потік обробки більше не повинен виконуватися
            this.exitEvent[0].Set();

            // Примусово знімаємо з паузи
            this.executeEvent[0].Set();

            // Знімаємо з очікування в циклі
            this.wakeUpEvent[0].Set();
        }

        /// <summary>
        /// Постановка потоку обробки на паузу
        /// </summary>
        public void Pause()
        {
            // Ставимо на паузу
            this.executeEvent[0].Reset();

            // Знімаємо з очікування в циклі
            this.wakeUpEvent[0].Set();
        }

        /// <summary>
        /// Зняття потоку обробки з паузи
        /// </summary>
        public void Continue()
        {
            // Знімаємо обробку с паузи
            this.executeEvent[0].Set();
        }

        #endregion Public Operations

        #region Private Operations

        /// <summary>
        /// Обчислення й запис у кінець файлів значення CRC-64
        /// </summary>
        private void WriteCRC64()
        {
            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = (this.dataCount + this.eccCount) / 100;

            // Якщо модуль дорівнює нулю, то збільшуємо його до значення "1",
щоб
            // прогрес виводився на кожній ітерації (файл дуже маленький)
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }

            // Піддаємо обробці всі томи
            for (int volNum = 0; volNum < (this.dataCount + this.eccCount);
volNum++)
            {
```

```
                // Зчитуємо первісне ім'я файлу
                String fileName = this.fileName;

                // Одержуємо ім'я вихідного файлу в префіксній формі
                this.eFileNamer.Pack(ref fileName, volNum, this.dataCount,
this.eccCount, this.codecType);

                // Формуємо повне ім'я файлу
                fileName = this.path + fileName;

                // Робимо обчислення CRC-64 для кожного файлу
                if (this.eFileIntegrityCheck.StartToWriteCRC64(fileName, true))
                {
                    // Цикл очікування завершення обробки файлу
                    while (true)
                    {
                        // Якщо не виявили встановленої події "executeEvent",
                        // те користувач хоче, щоб ми поставили обробку на паузу
-
                        if (!ManualResetEvent.WaitAll(this.executeEvent, 0,
false))
                        {
                            //...припиняємо роботу контрольованого алгоритму...
                            this.eFileIntegrityCheck.Pause();

                            //...програма переходить у режим сна
                            ManualResetEvent.WaitAll(this.executeEvent);

                            // А коли прокинулися, указуємо, що обробка повинна
тривати
                            this.eFileIntegrityCheck.Continue();
                        }

                        // Чекаємо кожне з перерахованих подій...
                        int eventIdx = ManualResetEvent.WaitAny(new
ManualResetEvent[] { this.wakeUpEvent[0], this.exitEvent[0],
this.eFileIntegrityCheck.FinishedEvent[0] });

                        //...якщо одержали сигнал до того, щоб прокинутися -
                        // переходимо на нову ітерацію, тому що прокидаємося
                        // перед постановкою на паузу...
                        if (eventIdx == 0)
                        {
                            //...попередньо скинувши подію, що змусила нас
прокинутися
                            this.wakeUpEvent[0].Reset();

                            continue;
                        }

                        //...якщо одержали сигнал до виходу з обробки...
                        if (eventIdx == 1)
                        {
                            //...зупиняємо контрольований алгоритм
                            this.eFileIntegrityCheck.Stop();

                            // Указуємо на те, що обробка була перервана
                            this.processedOK = false;

                            // Активуємо індикатор актуального стану змінних-
членів
                            this.finished = true;

                            // Установлюємо подію завершення обробки
                            this.finishedEvent[0].Set();

                            return;
                        }
```

```
                        //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                        if (eventIdx == 2)
                        {
                            //...виходимо із циклу очікування завершення (цього
й чекали в while(true)!)
                            break;
                        }

                    } // while(true)

                } else
                {
                    // Скидаємо прапор коректності результату
                    this.processedOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }

                // У зв'язку із закриттям великої кількості файлових потоків
                // необхідно дочекатися запису змін, внесених потоком
                // кодування в тіло класу. Потік уже не працює, але
                // установлена їм Булевська властивість, можливо, ще
                // "не виявилося"
                for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                {
                    if (!this.eFileIntegrityCheck.Finished)
                    {
                        Thread.Sleep((int)WaitTime.MinWaitTime);

                    } else
                    {
                        break;
                    }
                }

                // Якщо цикли очікування закриття файлових потоків не привели до
бажаного
                // результату - це помилка
                if (!this.eFileIntegrityCheck.ProcessedOK)
                {
                    // Указуємо на те, що обробка не була завершена коректно
                    this.processedOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }

                // Виводимо прогрес обробки
                if (
                        ((volNum % progressMod1) == 0)
                     &&
                        (OnUpdateFileAnalyzeProgress != null)
                    )
                {
                    OnUpdateFileAnalyzeProgress(((double)(volNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                }
```

```
                // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                // буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Указуємо на те, що обробка була перервана
                    this.processedOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }
            }

            // Повідомляємо про закінчення процесу обробки
            if (OnFileAnalyzeFinish != null)
            {
                OnFileAnalyzeFinish();
            }

            // Повідомляємо, що обробка пройшла коректно
            this.processedOK = true;

            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;

            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }

          /// <summary>
        /// Обчислення й перевірка значення CRC-64, записаного наприкінці файлу
        /// </summary>
        private void AnalyzeCRC64()
        {
            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = (this.dataCount + this.eccCount) / 100;

            // Якщо модуль дорівнює нулю, то збільшуємо його до значення "1",
щоб
            // прогрес виводився на кожній ітерації (файл дуже маленький)
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }

            // Виділяємо пам'ять під "volList"
            this.volList = new int[this.dataCount];

            // Виділяємо пам'ять під "altEccList"
            int[] altEccList = new int[this.eccCount];

            // Індекс у масиві томів
            int volListIdx = 0;

            // Індекс у масиві томів для відновлення
            int altEccListIdx = 0;

            // Лічильник кількості ушкоджених основних томів
```

```
            int dataVolMissCount = 0;

            // Лічильник кількості знайдених томів для відновлення
            int eccVolPresentCount = 0;

            // Ім'я файлу для обробки
            String fileName;

            // Піддаємо перевірці всі основні томи
            for (int dataNum = 0; dataNum < this.dataCount; dataNum++)
            {
                // Спочатку припускаємо, що поточний том ушкоджено
                bool dataVolIsOK = false;

                // Зчитуємо первісне ім'я файлу
                fileName = this.fileName;

                // Одержуємо ім'я вихідного файлу в префіксній формі
                this.eFileNamer.Pack(ref fileName, dataNum, this.dataCount,
this.eccCount, this.codecType);

                // Формуємо повне ім'я файлу
                fileName = this.path + fileName;

                // Якщо вихідний файл існує...
                if (File.Exists(fileName))
                {
                    // Якщо не використовується швидке добування - перевіряємо
на цілісність
                    // CRC-64, інакше думаємо, що все корректно (цілісність тому
беремо
                    // по факті його наявності)
                    if (!this.fastExtraction)
                    {
                        //...- робимо його перевірку
                        if (this.eFileIntegrityCheck.StartToCheckCRC64(fileName,
true))
                        {
                            // Цикл очікування завершення обробки файлу
                            while (true)
                            {
                                // Якщо не виявили встановленої події
"executeEvent",
                                // те користувач хоче, щоб ми поставили обробку
на паузу -
                                if (!ManualResetEvent.WaitAll(this.executeEvent,
0, false))
                                {
                                    //...припиняємо роботу контрольованого
алгоритму...
                                    this.eFileIntegrityCheck.Pause();

                                    //...програма переходить у режим сна
                                    ManualResetEvent.WaitAll(this.executeEvent);

                                    // А коли прокинулися, указуємо, що обробка
повинна тривати
                                    this.eFileIntegrityCheck.Continue();
                                }

                                // Чекаємо кожне з перерахованих подій...
                                int eventIdx = ManualResetEvent.WaitAny(new
ManualResetEvent[] { this.wakeUpEvent[0], this.exitEvent[0],
this.eFileIntegrityCheck.FinishedEvent[0] });

                                //...якщо одержали сигнал до того, щоб
прокинутися -
                                // переходимо на нову ітерацію, тому що
прокидаємося
```

```
                                // перед постановкою на паузу...
                                if (eventIdx == 0)
                                {
                                    //...попередньо скинувши подію, що змусила
нас прокинутися
                                    this.wakeUpEvent[0].Reset();

                                    continue;
                                }

                                //...якщо одержали сигнал до виходу з обробки...
                                if (eventIdx == 1)
                                {
                                    //...зупиняємо контрольований алгоритм
                                    this.eFileIntegrityCheck.Stop();

                                    // Указуємо на те, що обробка була перервана
                                    this.processedOK = false;

                                    // Активуємо індикатор актуального стану
змінних-членів
                                    this.finished = true;

                                    // Установлюємо подію завершення обробки
                                    this.finishedEvent[0].Set();

                                    return;
                                }

                                //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                                if (eventIdx == 2)
                                {
                                    //...виходимо із циклу очікування завершення
(цього й чекали в while(true)!)
                                    break;
                                }

                            } // while(true)

                        } else
                        {
                            // Скидаємо прапор коректності результату
                            this.processedOK = false;

                            // Активуємо індикатор актуального стану змінних-
членів
                            this.finished = true;

                            // Установлюємо подію завершення обробки
                            this.finishedEvent[0].Set();

                            return;
                        }

                        // У зв'язку із закриттям великої кількості файлових
потоків
                        // необхідно дочекатися запису змін, внесених потоком
                        // кодування в тіло класу. Потік уже не працює, але
                        // установлене їм Булевська властивість, можливо, ще
                        // "не виявилося"
                        for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                        {
                            if (!this.eFileIntegrityCheck.Finished)
                            {
                                Thread.Sleep((int)WaitTime.MinWaitTime);

                            } else
                            {
```

```
                                break;
                            }
                        }

                        // Указуємо, що основний том коректний
                        if (this.eFileIntegrityCheck.ProcessedOK)
                        {
                            dataVolIsOK = true;
                        }

                    } else
                    {
                        // Указуємо, що основний том коректний
                        dataVolIsOK = true;
                    }

                    // Виводимо прогрес обробки
                    if (
                            ((dataNum % progressMod1) == 0)
                     &&
                            (OnUpdateFileAnalyzeProgress != null)
                        )
                    {
                        OnUpdateFileAnalyzeProgress(((double)(dataNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                    }

                    // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                    // буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        // Указуємо на те, що обробка була перервана
                        this.processedOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                // Якщо даний основний том не ушкоджений, записуємо його в
"volList",
                // а інакше збільшуємо лічильник ушкоджених томів і ставимо на
місце
                // номера тому значення "-1", що вкаже на необхідність
підстановки
                // тому для відновлення
                if (dataVolIsOK)
                {
                    this.volList[volListIdx++] = dataNum;

                } else
                {
                    this.volList[volListIdx++] = -1;

                    // Збільшуємо лічильник кількості ушкоджених основних томів
                    dataVolMissCount++;
                }
            }

            // Тепер, коли знаємо кількість ушкоджених основних томів,
```

```
            // потрібно просканувати всі файли для відновлення, і визначити
            // необхідну їхню частину в список томів, а "надлишок" помістити в
            // список альтренативних томів для відновлення
            for (int eccNum = this.dataCount; eccNum < (this.dataCount +
this.eccCount); eccNum++)
            {
                // Спочатку припускаємо, що поточний том ушкоджено
                bool eccVolIsOK = false;

                // Зчитуємо первісне ім'я файлу
                fileName = this.fileName;

                // Одержуємо ім'я вихідного файлу в префіксній формі
                this.eFileNamer.Pack(ref fileName, eccNum, this.dataCount,
this.eccCount, this.codecType);

                // Формуємо повне ім'я файлу
                fileName = this.path + fileName;

                // Якщо вихідний файл існує...
                if (File.Exists(fileName))
                {
                    // Якщо не використовується швидке добування - перевіряємо
на цілісність
                    // CRC-64, інакше думаємо, що все корректно (цілісність тому
беремо
                    // по факту його наявності)
                    if (!this.fastExtraction)
                    {
                        //...- робимо його перевірку
                        if (this.eFileIntegrityCheck.StartToCheckCRC64(fileName,
true))
                        {
                            // Цикл очікування завершення обробки файлу
                            while (true)
                            {
                                // Якщо не виявили встановленої події
"executeEvent",
                                // то користувач хоче, щоб ми поставили обробку
на паузу -
                                if (!ManualResetEvent.WaitAll(this.executeEvent,
0, false))
                                {
                                    //...припиняємо роботу контрольованого
алгоритму...
                                    this.eFileIntegrityCheck.Pause();

                                    //...програма переходить у режим сна
                                    ManualResetEvent.WaitAll(this.executeEvent);

                                    // А коли прокинулися, указуємо, що обробка
повинна тривати
                                    this.eFileIntegrityCheck.Continue();
                                }

                                // Чекаємо кожне з перерахованих подій...
                                int eventIdx = ManualResetEvent.WaitAny(new
ManualResetEvent[] { this.wakeUpEvent[0], this.exitEvent[0],
this.eFileIntegrityCheck.FinishedEvent[0] });

                                //...якщо одержали сигнал до того, щоб
прокинутися -
                                // переходимо на нову ітерацію, тому що
прокидаємося
                                // перед постановкою на паузу...
                                if (eventIdx == 0)
                                {
                                    //...попередньо скинувши подію, що змусила
нас прокинутися
```

```
                                this.wakeUpEvent[0].Reset();

                                continue;
                            }

                            //...якщо одержали сигнал до виходу з обробки...
                            if (eventIdx == 1)
                            {
                                //...зупиняємо контрольований алгоритм
                                this.eFileIntegrityCheck.Stop();

                                // Указуємо на те, що обробка була перервана
                                this.processedOK = false;

                                // Активуємо індикатор актуального стану
змінних-членів
                                this.finished = true;

                                // Установлюємо подію завершення обробки
                                this.finishedEvent[0].Set();

                                return;
                            }

                            //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                            if (eventIdx == 2)
                            {
                                //...виходимо із циклу очікування завершення
(цього й чекали в while(true)!)
                                break;
                            }

                        } // while(true)

                    } else
                    {
                        // Скидаємо прапор коректності результату
                        this.processedOK = false;

                        // Активуємо індикатор актуального стану змінних-
членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }

                    // У зв'язку із закриттям великої кількості файлових
потоків
                    // необхідно дочекатися запису змін, внесених потоком
                    // кодування в тіло класу. Потік уже не працює, але
                    // установлена їм Булевська властивість, можливо, ще
                    // "не виявилася"
                    for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                    {
                        if (!this.eFileIntegrityCheck.Finished)
                        {
                            Thread.Sleep((int)WaitTime.MinWaitTime);

                        } else
                        {
                            break;
                        }
                    }

                    // Указуємо, що том для відновлення коректний
```

```
                        if (this.eFileIntegrityCheck.ProcessedOK)
                        {
                            eccVolIsOK = true;
                        }

                    } else
                    {
                        // Указуємо, що том для відновлення коректний
                        eccVolIsOK = true;
                    }

                    // Виводимо прогрес обробки
                    if (
                            ((eccNum % progressMod1) == 0)
                         &&
                            (OnUpdateFileAnalyzeProgress != null)
                        )
                    {
                        OnUpdateFileAnalyzeProgress(((double)(eccNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                    }

                    // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                    // буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        // Указуємо на те, що обробка була перервана
                        this.processedOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                // Якщо том для відновлення гарний...
                if (eccVolIsOK)
                {
                    //...- додаємо його в список
                    altEccList[altEccListIdx++] = eccNum;

                    // Збільшуємо лічильник кількості томів для відновлення
                    eccVolPresentCount++;

                } else
                {
                    //...а інакше вказуємо, що том ушкоджено
                    altEccList[altEccListIdx++] = -1;
                }
            }

            // Якщо значення лічильника кількості коректних томів для
відновлення збігається
            // зі значенням лічильника томів для відновлення конфігурації - всі
томи для
            // відновлення є неушкодженими
            if (eccVolPresentCount == this.eccCount)
            {
                this.allEccVolsOK = true;
            }
```

```
            // Виводимо статистику ушкоджень
            if (OnGetDamageStat != null)
            {
                // Обчислюємо загальний відсоток ушкоджень (суму ушкоджень
                // основних томів і томів для відновлення ділимо на загальну
кількість томів)
                double percOfDamage = ((double)(dataVolMissCount +
(this.eccCount - eccVolPresentCount)) / (double)(this.dataCount +
this.eccCount)) * 100;

                // Обчислюємо відсоток "" альтернативних томів, щовижили, для
відновлення
                // Альтернативні томи - це спочатку ті томи, які не планується
використовувати для відновлення
                double percOfAltEcc = ((double)(eccVolPresentCount -
dataVolMissCount) / (double)this.eccCount) * 100;

                // Виводимо статистику ушкоджень
                OnGetDamageStat(percOfDamage, percOfAltEcc);
            }

            // Якщо немає ушкоджених основних томів, просто виходимо
            if (dataVolMissCount == 0)
            {
                // Повідомляємо про закінчення процесу обробки
                if (OnFileAnalyzeFinish != null)
                {
                    OnFileAnalyzeFinish();
                }

                // Указуємо на те, що дані не ушкоджені
                this.processedOK = true;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return;
            }

            // Якщо ми не зможемо відновити ушкодження...
            if (eccVolPresentCount < dataVolMissCount)
            {
                //...указуємо на те, що дані не можуть бути відновлені
                this.processedOK = false;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return;
            }

            // Переміщаємося на початок списку альтернативних томів для
відновлення
            altEccListIdx = 0;

            // Тепер пробігаємося по вектору "volList", і замість кожного зі
значень "-1"
            // підставляємо чергове значення зі знайденого діапазону
            for (int i = 0; i < this.dataCount; i++)
            {
                if (this.volList[i] == -1)
                {
                    // Пробігаємося по векторі томів для відновлення,
```

```
                        // зупиняючись на коректному томі для відновлення
                        while (altEccList[altEccListIdx] == -1)
                        {
                            altEccListIdx++;
                        }

                        // Підставляємо на місце ушкодженого основного тому
                        // том для відновлення,...
                        this.volList[i] = altEccList[altEccListIdx];

                        //...забираючи використаний том зі списку альтернативних
                        altEccList[altEccListIdx] = -1;
                    }
                }

                // Повідомляємо про закінчення процесу обробки
                if (OnFileAnalyzeFinish != null)
                {
                    OnFileAnalyzeFinish();
                }

                // Повідомляємо, що обробка пройшла коректно
                this.processedOK = true;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();
            }

            #endregion Private Operations
        }
    }
```

Файл MainForm.cs - головне вікно програми

```
namespace Data_Center
{
    partial class MainForm
    {
        /// <summary>
        ///
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        ///
        /// </summary>
        /// <param name="disposing">правда, якщо керуючі ресурси повині бути
розташовані, неправда в іншому випадку.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Викликаємо метод для підтримки інтерфейсу
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.Windows.Forms.TreeNode treeNode1 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode2 = new
System.Windows.Forms.TreeNode("Documents and Settings", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode1});
            System.Windows.Forms.TreeNode treeNode3 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode4 = new
System.Windows.Forms.TreeNode("Завантаження", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode3});
            System.Windows.Forms.TreeNode treeNode5 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode6 = new
System.Windows.Forms.TreeNode("Program Files", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode5});
            System.Windows.Forms.TreeNode treeNode7 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode8 = new
System.Windows.Forms.TreeNode("RapidDriver", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode7});
            System.Windows.Forms.TreeNode treeNode9 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode10 = new
System.Windows.Forms.TreeNode("Server", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode9});
            System.Windows.Forms.TreeNode treeNode11 = new
System.Windows.Forms.TreeNode("");
```

```
            System.Windows.Forms.TreeNode treeNode12 = new
System.Windows.Forms.TreeNode("WINDOWS", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode11});
            System.Windows.Forms.TreeNode treeNode13 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode14 = new
System.Windows.Forms.TreeNode("WINDOWS.0", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode13});
            System.Windows.Forms.TreeNode treeNode15 = new
System.Windows.Forms.TreeNode("SYSTEM2 (C:)", 23, 24, new
System.Windows.Forms.TreeNode[] {
            treeNode2,
            treeNode4,
            treeNode6,
            treeNode8,
            treeNode10,
            treeNode12,
            treeNode14});
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(MainForm));
            this.menuStrip = new System.Windows.Forms.MenuStrip();
            this.fileToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.instrumentToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.adjustmentToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.encodingfilterToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.quickextractionToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.helpToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.separatorToolStripMenuItem = new
System.Windows.Forms.ToolStripSeparator();
            this.aboutToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.coderConfigGroupBox = new System.Windows.Forms.GroupBox();
            this.redundancyGroupBox = new System.Windows.Forms.GroupBox();
            this.redundancyMacTrackBar = new
EConTech.Windows.MACUI.MACTrackBar();
            this.allVolCountGroupBox = new System.Windows.Forms.GroupBox();
            this.allVolCountMacTrackBar = new
EConTech.Windows.MACUI.MACTrackBar();
            this.toolTip = new System.Windows.Forms.ToolTip(this.components);
            this.browser = new FileBrowser.Browser();
            this.repairButton = new System.Windows.Forms.Button();
            this.testButton = new System.Windows.Forms.Button();
            this.recoverButton = new System.Windows.Forms.Button();
            this.protectButton = new System.Windows.Forms.Button();
            this.exitToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.testspeedToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.menuStrip.SuspendLayout();
            this.coderConfigGroupBox.SuspendLayout();
            this.redundancyGroupBox.SuspendLayout();
            this.allVolCountGroupBox.SuspendLayout();
            this.SuspendLayout();
            //
            // menuStrip
            //
            this.menuStrip.Items.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.fileToolStripMenuItem,
            this.instrumentToolStripMenuItem,
            this.adjustmentToolStripMenuItem,
```

```
            this.helpToolStripMenuItem});
            this.menuStrip.LayoutStyle =
System.Windows.Forms.ToolStripLayoutStyle.Flow;
            this.menuStrip.Location = new System.Drawing.Point(0, 0);
            this.menuStrip.Name = "menuStrip";
            this.menuStrip.Size = new System.Drawing.Size(986, 21);
            this.menuStrip.TabIndex = 0;
            this.menuStrip.Text = "menuStrip";
            //
            // fileToolStripMenuItem
            //
            this.fileToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.exitToolStripMenuItem});
            this.fileToolStripMenuItem.Name = "файлToolStripMenuItem";
            this.fileToolStripMenuItem.Size = new System.Drawing.Size(45, 17);
            this.fileToolStripMenuItem.Text = "Файл";
            //
            // instrumentsToolStripMenuItem
            //
            this.instrumentToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.testspeedToolStripMenuItem});
            this.instrumentToolStripMenuItem.Name =
"instrumentsToolStripMenuItem";
            this.instrumentToolStripMenuItem.Size = new System.Drawing.Size(82,
17);
            this.instrumentToolStripMenuItem.Text = "Інструменти";
            //
            // adjustmentToolStripMenuItem
            //
            this.adjustmentToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.encodingfilterToolStripMenuItem,
            this.quickextractionToolStripMenuItem});
            this.adjustmentToolStripMenuItem.Name =
"adjustmentToolStripMenuItem";
            this.adjustmentToolStripMenuItem.Size = new System.Drawing.Size(74,
17);
            this.adjustmentToolStripMenuItem.Text = "Параметри";
            //
            // encodingfilterToolStripMenuItem
            //
            this.encodingfilterToolStripMenuItem.ImageScaling =
System.Windows.Forms.ToolStripItemImageScaling.None;
            this.encodingfilterToolStripMenuItem.Name =
"encodingfilterToolStripMenuItem";
            this.encodingfilterToolStripMenuItem.Size = new
System.Drawing.Size(216, 22);
            this.encodingfilterToolStripMenuItem.Text = "Шифруючий фільтр";
            this.encodingfilterToolStripMenuItem.Click += new
System.EventHandler(this.encodingfilterToolStripMenuItem_Click);
            //
            // helpToolStripMenuItem
            //
            this.helpToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.separatorToolStripMenuItem,
            this.aboutToolStripMenuItem});
            this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
            this.helpToolStripMenuItem.Size = new System.Drawing.Size(60, 17);
            this.helpToolStripMenuItem.Text = "Довідка";
            //
            // separatorToolStripMenuItem
            //
            this.separatorToolStripMenuItem.Name = "separatorToolStripMenuItem";
            this.separatorToolStripMenuItem.Size = new System.Drawing.Size(163,
6);
            //
```

```
            // aboutToolStripMenuItem
            //
            this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
            this.aboutToolStripMenuItem.Size = new System.Drawing.Size(166, 22);
            this.aboutToolStripMenuItem.Text = "Про програму...";
            this.aboutToolStripMenuItem.Click += new
System.EventHandler(this.aboutToolStripMenuItem_Click);
            //
            // coderConfigGroupBox
            //
            this.coderConfigGroupBox.Controls.Add(this.redundancyGroupBox);
            this.coderConfigGroupBox.Controls.Add(this.allVolCountGroupBox);
            this.coderConfigGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.coderConfigGroupBox.Location = new System.Drawing.Point(414,
26);
            this.coderConfigGroupBox.Name = "coderConfigGroupBox";
            this.coderConfigGroupBox.Size = new System.Drawing.Size(561, 98);
            this.coderConfigGroupBox.TabIndex = 5;
            this.coderConfigGroupBox.TabStop = false;
            this.coderConfigGroupBox.Text = "Конфігурація системи";
            //
            // redundancyGroupBox
            //
            this.redundancyGroupBox.Controls.Add(this.redundancyMacTrackBar);
            this.redundancyGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.redundancyGroupBox.Location = new System.Drawing.Point(286,
21);
            this.redundancyGroupBox.Name = "redundancyGroupBox";
            this.redundancyGroupBox.Size = new System.Drawing.Size(264, 65);
            this.redundancyGroupBox.TabIndex = 4;
            this.redundancyGroupBox.TabStop = false;
            this.redundancyGroupBox.Text = "Надлишковість кодування";
            //
            // allVolCountGroupBox
            //
            this.allVolCountGroupBox.Controls.Add(this.allVolCountMacTrackBar);
            this.allVolCountGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.allVolCountGroupBox.Location = new System.Drawing.Point(12,
21);
            this.allVolCountGroupBox.Name = "allVolCountGroupBox";
            this.allVolCountGroupBox.Size = new System.Drawing.Size(264, 65);
            this.allVolCountGroupBox.TabIndex = 3;
            this.allVolCountGroupBox.TabStop = false;
            this.allVolCountGroupBox.Text = "Довжина коду";
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // redundancyMacTrackBar
            //
            this.redundancyMacTrackBar.BackColor =
System.Drawing.Color.Transparent;
            this.redundancyMacTrackBar.BorderColor =
System.Drawing.SystemColors.ActiveBorder;
            this.redundancyMacTrackBar.Font = new System.Drawing.Font("Verdana",
8.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.redundancyMacTrackBar.ForeColor =
System.Drawing.Color.FromArgb(((int)(((byte)(123)))), ((int)(((byte)(125)))),
((int)(((byte)(123)))));
            this.redundancyMacTrackBar.IndentHeight = 6;
```

```
            this.redundancyMacTrackBar.Location = new System.Drawing.Point(6,
24);
            this.redundancyMacTrackBar.Maximum = 199;
            this.redundancyMacTrackBar.Minimum = 0;
            this.redundancyMacTrackBar.Name = "redundancyMacTrackBar";
            this.redundancyMacTrackBar.Size = new System.Drawing.Size(252, 28);
            this.redundancyMacTrackBar.TabIndex = 6;
            this.redundancyMacTrackBar.TextTickStyle =
System.Windows.Forms.TickStyle.None;
            this.redundancyMacTrackBar.TickColor =
System.Drawing.Color.FromArgb(((int)(((byte)(148)))), ((int)(((byte)(146)))),
((int)(((byte)(148)))));
            this.redundancyMacTrackBar.TickHeight = 4;
            this.redundancyMacTrackBar.TickStyle =
System.Windows.Forms.TickStyle.None;
            this.redundancyMacTrackBar.TrackerColor =
System.Drawing.Color.FromArgb(((int)(((byte)(24)))), ((int)(((byte)(130)))),
((int)(((byte)(198)))));
            this.redundancyMacTrackBar.TrackerSize = new System.Drawing.Size(10,
16);
            this.redundancyMacTrackBar.TrackLineColor =
System.Drawing.Color.FromArgb(((int)(((byte)(90)))), ((int)(((byte)(93)))),
((int)(((byte)(90)))));
            this.redundancyMacTrackBar.TrackLineHeight = 3;
            this.redundancyMacTrackBar.Value = 19;
            this.redundancyMacTrackBar.ValueChanged += new
EConTech.Windows.MACUI.ValueChangedHandler(this.redundancyMacTrackBar_ValueChang
ed);
            this.redundancyMacTrackBar.MouseUp += new
System.Windows.Forms.MouseEventHandler(this.redundancyMacTrackBar_MouseUp);
            //
            // allVolCountMacTrackBar
            //
            this.allVolCountMacTrackBar.BackColor =
System.Drawing.Color.Transparent;
            this.allVolCountMacTrackBar.BorderColor =
System.Drawing.SystemColors.ActiveBorder;
            this.allVolCountMacTrackBar.Font = new
System.Drawing.Font("Verdana", 8.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.allVolCountMacTrackBar.ForeColor =
System.Drawing.Color.FromArgb(((int)(((byte)(123)))), ((int)(((byte)(125)))),
((int)(((byte)(123)))));
            this.allVolCountMacTrackBar.IndentHeight = 6;
            this.allVolCountMacTrackBar.Location = new System.Drawing.Point(6,
24);
            this.allVolCountMacTrackBar.Maximum = 15;
            this.allVolCountMacTrackBar.Minimum = 0;
            this.allVolCountMacTrackBar.Name = "allVolCountMacTrackBar";
            this.allVolCountMacTrackBar.Size = new System.Drawing.Size(252, 28);
            this.allVolCountMacTrackBar.TabIndex = 5;
            this.allVolCountMacTrackBar.TextTickStyle =
System.Windows.Forms.TickStyle.None;
            this.allVolCountMacTrackBar.TickColor =
System.Drawing.Color.FromArgb(((int)(((byte)(148)))), ((int)(((byte)(146)))),
((int)(((byte)(148)))));
            this.allVolCountMacTrackBar.TickHeight = 4;
            this.allVolCountMacTrackBar.TickStyle =
System.Windows.Forms.TickStyle.None;
            this.allVolCountMacTrackBar.TrackerColor =
System.Drawing.Color.FromArgb(((int)(((byte)(24)))), ((int)(((byte)(130)))),
((int)(((byte)(198)))));
            this.allVolCountMacTrackBar.TrackerSize = new
System.Drawing.Size(10, 16);
            this.allVolCountMacTrackBar.TrackLineColor =
System.Drawing.Color.FromArgb(((int)(((byte)(90)))), ((int)(((byte)(93)))),
((int)(((byte)(90)))));
            this.allVolCountMacTrackBar.TrackLineHeight = 3;
            this.allVolCountMacTrackBar.Value = 2;
```

```
            this.allVolCountMacTrackBar.ValueChanged += new
EConTech.Windows.MACUI.ValueChangedHandler(this.allVolCountMacTrackBar_ValueChan
ged);
            this.allVolCountMacTrackBar.MouseUp += new
System.Windows.Forms.MouseEventHandler(this.allVolCountMacTrackBar_MouseUp);
            //
            // browser
            //
            this.browser.AutoValidate =
System.Windows.Forms.AutoValidate.EnablePreventFocusChange;
            this.browser.ListViewMode = System.Windows.Forms.View.List;
            this.browser.Location = new System.Drawing.Point(12, 131);
            this.browser.Name = "browser";
            treeNode1.Name = "";
            treeNode1.Text = "";
            treeNode2.ImageIndex = 18;
            treeNode2.Name = "backreg";
            treeNode2.SelectedImageIndex = 20;
            treeNode2.Text = "backreg";
            treeNode3.Name = "";
            treeNode3.Text = "";
            treeNode4.ImageIndex = 18;
            treeNode4.Name = "";
            treeNode4.SelectedImageIndex = 20;
            treeNode4.Text = "Code_Rid_Solomon_";
            treeNode5.Name = "";
            treeNode5.Text = "";
            treeNode6.ImageIndex = 18;
            treeNode6.Name = "Documents and Settings";
            treeNode6.SelectedImageIndex = 20;
            treeNode6.Text = "Documents and Settings";
            treeNode7.Name = "";
            treeNode7.Text = "";
            treeNode8.ImageIndex = 18;
            treeNode8.Name = "Завантаження";
            treeNode8.SelectedImageIndex = 20;
            treeNode8.Text = "Завантаження";
            treeNode9.Name = "";
            treeNode9.Text = "";
            treeNode10.ImageIndex = 18;
            treeNode10.Name = "Inprise";
            treeNode10.SelectedImageIndex = 20;
            treeNode10.Text = "Inprise";
            treeNode11.Name = "";
            treeNode11.Text = "";
            treeNode12.ImageIndex = 18;
            treeNode12.Name = "Program Files";
            treeNode12.SelectedImageIndex = 20;
            treeNode12.Text = "Program Files";
            treeNode13.ImageIndex = 33;
            treeNode13.Name = "Recycled";
            treeNode13.SelectedImageIndex = 34;
            treeNode13.Text = "Recycled";
            treeNode14.ImageIndex = 18;
            treeNode14.Name = "КОРЗИНА";
            treeNode14.SelectedImageIndex = 20;
            treeNode14.Text = "КОРЗИНА";
            treeNode15.ImageIndex = 18;
            treeNode15.Name = "Системна інформація";
            treeNode15.SelectedImageIndex = 20;
            treeNode15.Text = "Системна інформація";
            treeNode16.ImageIndex = 18;
            treeNode16.Name = "temp";
            treeNode16.SelectedImageIndex = 20;
            treeNode16.Text = "temp";
            treeNode17.Name = "";
            treeNode17.Text = "";
            treeNode18.ImageIndex = 18;
            treeNode18.Name = "WINDOWS";
```

```
            treeNode18.SelectedImageIndex = 20;
            treeNode18.Text = "WINDOWS";
            treeNode19.ImageIndex = 23;
            treeNode19.Name = "Локальный диск (C:)";
            treeNode19.SelectedImageIndex = 24;
            treeNode19.Text = "Локальный диск (C:)";
            this.browser.SelectedNode = treeNode19;
            this.browser.ShowFoldersButton = false;
            this.browser.ShowNavigationBar = false;
            this.browser.Size = new System.Drawing.Size(962, 432);
            this.browser.SplitterDistance = 398;
            this.browser.StartUpDirectoryOther = "C:\\";
            this.browser.TabIndex = 0;
            this.browser.Load += new System.EventHandler(this.browser_Load);
            //
            // testButton
            //
            this.testButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.testButton.Image =
global::RecoveryData.Properties.Resources.table_sql_view32451;
            this.testButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.testButton.Location = new System.Drawing.Point(111, 27);
            this.testButton.Name = "testButton";
            this.testButton.Size = new System.Drawing.Size(100, 97);
            this.testButton.TabIndex = 4;
            this.testButton.Text = "Перевірка цілісності";
            this.testButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.testButton.UseVisualStyleBackColor = true;
            this.testButton.Click += new
System.EventHandler(this.testButton_Click);
            //
            // repairButton
            //
            this.repairButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.repairButton.Image =
global::RecoveryData.Properties.Resources.medical_bag465471;
            this.repairButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.repairButton.Location = new System.Drawing.Point(210, 27);
            this.repairButton.Name = "repairButton";
            this.repairButton.Size = new System.Drawing.Size(100, 97);
            this.repairButton.TabIndex = 3;
            this.repairButton.Text = "Відновлення цілісності";
            this.repairButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.repairButton.UseVisualStyleBackColor = true;
            this.repairButton.Click += new
System.EventHandler(this.repairButton_Click);
            //
            // recoverButton
            //
            this.recoverButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.recoverButton.Image =
global::RecoveryData.Properties.Resources.redo6786987;
            this.recoverButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.recoverButton.Location = new System.Drawing.Point(309, 27);
            this.recoverButton.Name = "recoverButton";
            this.recoverButton.Size = new System.Drawing.Size(100, 97);
            this.recoverButton.TabIndex = 2;
            this.recoverButton.Text = "Читання даних з диску";
            this.recoverButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.recoverButton.UseVisualStyleBackColor = true;
            this.recoverButton.Click += new
System.EventHandler(this.recoverButton_Click);
            //
```

```
            // protectButton
            //
            this.protectButton.BackColor = System.Drawing.SystemColors.Control;
            this.protectButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.protectButton.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8.25F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.protectButton.Image =
global::RecoveryData.Properties.Resources.Database_1_64x64e65768;
            this.protectButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.protectButton.Location = new System.Drawing.Point(12, 27);
            this.protectButton.Name = "protectButton";
            this.protectButton.Size = new System.Drawing.Size(100, 97);
            this.protectButton.TabIndex = 1;
            this.protectButton.Text = "Запис даних на диск";
            this.protectButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.protectButton.UseVisualStyleBackColor = false;
            this.protectButton.Click += new
System.EventHandler(this.protectButton_Click);
            //
            // ToolStripMenuItem
            //
            this.ToolStripMenuItem.Image =
global::RecoveryData.Properties.Resources.Exit;
            this.ToolStripMenuItem.Name = "ToolStripMenuItem";
            this.ToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
            this.ToolStripMenuItem.Text = "Вихід";
            this.ToolStripMenuItem.Click += new
System.EventHandler(this.выходToolStripMenuItem_Click);
            //
            // ToolStripMenuItem
            //
            this.тестБыстродействияToolStripMenuItem.Image =
global::RecoveryData.Properties.Resources.StartBenchmark;
            this.тестБыстродействияToolStripMenuItem.ImageScaling =
System.Windows.Forms.ToolStripItemImageScaling.None;
            this.ToolStripMenuItem.Name = "ToolStripMenuItem";
            this.ToolStripMenuItem.Size = new System.Drawing.Size(161, 22);
            this.ToolStripMenuItem.Text = "Тест швидкодії";
            this.ToolStripMenuItem.Click += new
System.EventHandler(this.тестБыстродействияToolStripMenuItem_Click);
            //
            // MainForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(986, 576);
            this.Controls.Add(this.coderConfigGroupBox);
            this.Controls.Add(this.testButton);
            this.Controls.Add(this.repairButton);
            this.Controls.Add(this.recoverButton);
            this.Controls.Add(this.protectButton);
            this.Controls.Add(this.browser);
            this.Controls.Add(this.menuStrip);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.MainMenuStrip = this.menuStrip;
            this.MaximizeBox = false;
            this.Name = "MainForm";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Підвищення надійності зберігання даних на носіях
інформації";
            this.Load += new System.EventHandler(this.MainForm_Load);
```

```
            this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.MainForm_FormClosing);
            this.menuStrip.ResumeLayout(false);
            this.menuStrip.PerformLayout();
            this.coderConfigGroupBox.ResumeLayout(false);
            this.redundancyGroupBox.ResumeLayout(false);
            this.redundancyGroupBox.PerformLayout();
            this.allVolCountGroupBox.ResumeLayout(false);
            this.allVolCountGroupBox.PerformLayout();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.MenuStrip menuStrip;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.Button protectButton;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.ToolStripSeparator
separatorToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.Button repairButton;
        private System.Windows.Forms.Button testButton;
        private System.Windows.Forms.GroupBox coderConfigGroupBox;
        private System.Windows.Forms.GroupBox redundancyGroupBox;
        private System.Windows.Forms.GroupBox allVolCountGroupBox;
        private EConTech.Windows.MACUI.MACTrackBar allVolCountMacTrackBar;
        private EConTech.Windows.MACUI.MACTrackBar redundancyMacTrackBar;
        private System.Windows.Forms.ToolTip toolTip;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        internal FileBrowser.Browser browser;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.Button recoverButton;
    }
}
```

Файл Code_Rid_Solomon_Decoder.cs - декодер коду Ріда-Соломона

```
using System;
using System.Threading;

namespace Data_Center
{
    /// <summary>
    /// Клас декодера коду Ріда-Соломона
    /// </summary>
    public class Code_Rid_Solomon_Decoder : Code_Rid_Solomon_Base
    {
        #region Data

        // Масив булевських ознак "рядок матриці "FLog" тривіальна?"
        private bool[] FLogRowIsTrivial;

        // Список порядкових номерів наявних томів (нумерація з нуля)
        private int[] volList;

        #endregion Data

        #region Construction & Destruction

        /// <summary>
        /// Конструктор декодера за замовчуванням
        /// </summary>
        public Code_Rid_Solomon_Decoder()
        {
            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        /// <summary>
        /// Базовий конструктор декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        public Code_Rid_Solomon_Decoder(int dataCount, int eccCount, int[]
volList)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList,
(int)Code_Rid_Solomon_Type.Alternative);

            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        /// <summary>
        /// Розширений конструктор декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        /// <param name="codecType">Тип кодера коду Ріда-Соломона (по типу
матриці)</param>
        public Code_Rid_Solomon_Decoder(int dataCount, int eccCount, int[]
volList, int codecType)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList, codecType);
```

```
            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Установка конфігурації декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        /// <param name="codecType">Тип кодека кодера коду Ріда-Соломона (по
типу матриці)</param>
        /// <returns>Булевський прапор операції установки конфігурації</returns>
        public bool SetConfig(int dataCount, int eccCount, int[] volList, int
codecType)
        {
            int maxVolCount;

            // Установлюємо константи, що відповідають обраному режиму
            if (codecType == (int)Code_Rid_Solomon_Type.Dispersal)
            {
                maxVolCount = (int)Code_Rid_Solomon_Const.MaxVolCountDisp;

            } else
            {
                maxVolCount = (int)Code_Rid_Solomon_Const.MaxVolCountAlt;
            }

            // Перевіряємо конфігурацію на коректність
            if (
                    (dataCount > 0)
                 &&
                    (eccCount > 0)
                 &&
                    ((dataCount + eccCount) <= maxVolCount)
                 &&
                    (volList.Length >= dataCount)
                )
            {
                // Якщо основна конфігурація змінилася - сповіщаємо про це
                if (
                        (dataCount != this.n)
                     ||
                        (eccCount != this.m)
                     ||
                        (codecType != this.eCode_Rid_Solomon_Type)
                    )
                {
                    this.mainConfigChanged = true;
                }

                // Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eCode_Rid_Solomon_Type = codecType;

                // Також перераховуємо кількість ітерацій всіх стадій підготовки
                double n = this.n;
                double m = this.m;

                // Нормалізуємо значення для розрахунку, щоб уникнути
переповнення змінних
                NormalizeNM(ref n, ref m);
```

```
                // Кількість ітерацій, що відслідковуються прогресом, на першій
стадії
                // залежить від типу використовуваної матриці
                if (this.eCode_Rid_Solomon_Type ==
(int)Code_Rid_Solomon_Type.Alternative)
                {
                    this.iterOfFirstStage = m;

                } else
                {
                    this.iterOfFirstStage = ((n * m) * n) + (n * ((n + m) + (n *
(n + m))));
                }

                this.iterOfSecondStage = (n * (((n - 1) * (n - 1)) + (n * n)));

                // Виділяємо пам'ять під масив булевських ознак "рядок матриці
"FLog" тривіальна?"
                this.FLogRowIsTrivial = new bool[dataCount];

                // Зберігаємо список наявних томів
                this.volList = volList;

                this.configIsOK = true;

            } else
            {
                //...указуємо на помилку конфігурації
                this.configIsOK = false;
            }

            return this.configIsOK;
        }

        /// <summary>
        /// Метод множення матриці кодування на вхідний прологарифмований вектор
        /// </summary>
        /// <param name="dataEccLog">Прологарифмований вхідний вектор (дані +
ecc)</param>
        /// <param name="data">Вихідний вектор (відновлені вихідні дані)</param>
        /// <returns>Булевський прапор результату операції</returns>
        public bool Process(int[] dataEccLog, ref int[] data)
        {
            // Якщо кодер зконфігуровано некоректно, обробка неможлива!
            if (!this.configIsOK)
            {
                return false;
            }

            // Копіюємо покажчик на масив експонент для скорочення часу обігу
            int[] GF16Exp = this.eGF16.GFExpTable;

            // Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.n; i++)
            {
                // Якщо поточний рядок матриці не є тривіальної, робимо обробку
                if (!this.FLogRowIsTrivial[i])
                {
                    int mulSum = 0;     // Сума добутку рядка матриці на
стовпець
                    int i_n = i * this.n; // Зсув у масиві до елементів i-ой
рядка

                    for (int j = 0; j < this.n; j++)
                    {
                        mulSum ^= GF16Exp[this.FLog[i_n + j] + dataEccLog[j]];
                    }

                    data[i] = mulSum;
```

```
                } else
                {
                    data[i] = GF16Exp[dataEccLog[i]];
                }
            }

            return true;
        }

        #endregion Public Operations

        #region Private Operations

        /// <summary>
        /// Пошук матриці, зворотної до "FLog", методом Жорданових виключень
        /// (Дана модифікація методу може використовуватися тільки в тих
випадках,
        /// коли (-a) = (a), тому що через непотрібність пропущена стадія зміни
елементів),
        /// крім того, відсутній пошук ненульового розв'язного елемента (у
випадку
        /// роботи з матрицею Вандермонда наявність нуля на діагоналі - збій
кодека,
        /// тому ситуація з виявленням нуля сприймається винятково як помилка
        /// </summary>
        /// <returns>Булевський прапор результату операції</returns>
        private bool FInv()
        {
            // Обчислюємо розподіл відсотків ітерацій по стадіях для
            // коректної обробки відсотків
            double allStageIter   = this.iterOfFirstStage +
this.iterOfSecondStage;
            int percOfFirstStage  = (int)((100.0 * this.iterOfFirstStage)  /
allStageIter);
            int percOfSecondStage = (int)((100.0 * this.iterOfSecondStage) /
allStageIter);

            // Дана стадія повинна займати хоча б один відсоток
            // (для коректності розрахунків)
            if (percOfSecondStage == 0)
            {
                percOfSecondStage = 1;
            }

            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "k"
            int progressMod1 = this.n / percOfSecondStage;

            // Якщо модуль дорівнює нулю, то збільшуємо його до значення "1",
щоб
            // прогрес виводився на кожній ітерації
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }

            // Цикл вибору розв'язного елемента "pivot"
            for (int k = 0; k < this.n; ++k)
            {
                // Якщо даний рядок тривіальна - просто переходимо на нову
ітерацію
                if (this.FLogRowIsTrivial[k])
                {
                    continue;
                }

                // Зсув у масиві до елементів k-ой рядка
```

```
                int k_n = k * this.n;

                // Індекс розв'язного елемента
                int pivotIdx = k_n + k;

                // Витягаємо розв'язний елемент
                int pivot = this.FLog[pivotIdx];

                // Якщо розв'язний елемент дорівнює нулю - матриця не має
звороткої
                if (pivot == 0)
                {
                    //...указуємо на помилку конфігурації
                    this.configIsOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return false;
                }

                // Після добування розв'язного елемента поміщаємо на його місце
"1"
                this.FLog[pivotIdx] = 1;

                // Працюємо з рядками...
                for (int i = 0; i < this.n; i++)
                {
                    // Якщо перебуваємо на рядку розв'язного елемента -
переходимо
                    // на нову ітерацію
                    if (i == k)
                    {
                        continue;
                    }

                    // Зсув у масиві до елементів i-ой рядка
                    int i_n = i * this.n;

                    // Працюємо зі стовпцями...
                    for (int j = 0; j < this.n; j++)
                    {
                        // Якщо перебуваємо на стовпці розв'язного елемента -
переходимо
                        // на нову ітерацію...
                        if (j == k)
                        {
                            continue;
                        }

                        int idx = i_n + j;

                        //...а інакше робимо необхідні дії над матрицею:
                        // "A[i,j] = A[i,j] * pivot + A[i,k] * A[k,j]"
                        this.FLog[idx] = this.eGF16.Mul(this.FLog[idx], pivot) ^
this.eGF16.Mul(this.FLog[i_n + k], this.FLog[k_n + j]);
                    }
                }

                // Розподіл матриці на розв'язний елемент заміняємо множенням на
зворотний
                int pivotInv = this.eGF16.Inv(pivot);

                for (int i = 0; i < this.n; i++)
                {
                    // Зсув у масиві до елементів i-ой рядка
```

```
                    int i_n = (i * this.n);

                    for (int j = 0; j < this.n; j++)
                    {
                        int idx = i_n + j;

                        this.FLog[idx] = this.eGF16.Mul(this.FLog[idx],
pivotInv);
                    }
                }

                // Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((k % progressMod1) == 0)
                     &&
                        (OnUpdateCode_Rid_Solomon_MatrixFormingProgress != null)
                   )
                {
                    //...виводимо дані
                    OnUpdateCode_Rid_Solomon_MatrixFormingProgress((((double)(k
+ 1) / (double)this.n) * percOfSecondStage) + percOfFirstStage);
                }

                // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                // буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Указуємо, що декодер зконфігуровано некоректно
                    this.configIsOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return false;
                }
            }

            return true;
        }

        /// <summary>
        /// Обчислення логарифмів значень інвертованої матриці
        /// <summary>
        private void LogFCalc()
        {
            // Працюємо з рядками...
            for (int i = 0; i < this.n; i++)
            {
                // Зсув у масиві до елементів і-ой рядка
                int i_n = i * this.n;

                // Працюємо зі стовпцями...
                for (int j = 0; j < this.n; j++)
                {
                    int idx = i_n + j;

                    this.FLog[idx] = this.eGF16.Log(this.FLog[idx]);
                }
            }
        }

        /// <summary>
```

```
        /// Заповнення матриці "FLog" (матриці декодера) даними
        /// </summary>
        protected override void FillFLog()
        {
            // Якщо довжина вектора наявних томів менше кількості,
            // необхідного для відновлення...
            if (this.volList.Length < this.n)
            {
                //...указуємо на помилку конфігурації
                this.configIsOK = false;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return;
            }

            // Виділяємо пам'ять під матрицю "FLog"
            this.FLog = new int[this.n * this.n];

            // Вектор лічильників всіх томів...
            int[] allVolCount = new int[this.n + this.m];

            //...і вектор ecc-томів для "затикання" пробілів, створених
            // загубленими основними томами
            int[] eccVolToFix = new int[this.m];

            // Лічильник кількості стертих основних томів
            int dataVolMissCount = this.n;

            // Ініціалізуємо масив лічильників всіх томів
            for (int i = 0; i < (this.n + this.m); i++)
            {
                allVolCount[i] = 0;
            }

            // Проводимо аналіз складу представлених томів на предмет наявності
основних
            for (int i = 0; i < this.n; i++)
            {
                // Обчислюємо номер поточного тому
                int currVol = Math.Abs(this.volList[i]);

                // Якщо номер тому відповідає припустимому діапазону
                if (currVol < (this.n + this.m))
                {
                    ++allVolCount[currVol];

                    // Якщо поточний том є основним, фіксуємо даний факт
                    if (currVol < this.n)
                    {
                        ---idataVolMissCount;
                    }

                } else
                {
                    // Указуємо на помилку конфігурації
                    this.configIsOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
```

```
                }
            }

            // Перевіряємо лічильники томів на помилкове дублювання
            for (int i = 0; i < (this.n + this.m); i++)
            {
                // Якщо деякий том був зазначений більш ніж один раз...
                if (allVolCount[i] > 1)
                {
                    //...указуємо на помилку конфігурації
                    this.configIsOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }
            }

            // Якщо перевірка на несуперечність не виявила проблем, починаємо
            // формувати матрицю "FLog"

            // Якщо основна конфігурація змінилася...
            if (this.mainConfigChanged)
            {
                if (this.eCode_Rid_Solomon_Type ==
(int)Code_Rid_Solomon_Type.Dispersal)
                {
                    //...робимо формування дисперсної матриці "D"
                    if (!MakeDispersalMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }

                } else
                {
                    //...робимо формування альтернативного заповнення матриці
"A"
                    if (!MakeAlternativeMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                //...і скидаємо прапор
                this.mainConfigChanged = false;
            }
```

```
            // Для кожного загубленого основного тому шукаємо том для
відновлення
            for (int i = 0, j = 0; i < dataVolMissCount; i++)
            {
                // Рухаємося за списком томів доти, поки не знайдемо том для
                // відновлення для затикання "дірки" (основні томи мають номера
                // менше this.n (при нумерації з нуля!))
                while (this.volList[j] < this.n)
                {
                    j++;
                }

                // Зберігаємо номер тому для заміни загубленого основного тому
                eccVolToFix[i] = this.volList[j];

                j++; // j++ дозволяє перейти до наступного пошуку
            }

            // Працюємо по рядках матриці (в ідеалі, всі рядки повинні
заповнюватися
            // рядками з одиницею на головній діагоналі, що відповідає
відсутності
            // ушкоджень, але allVolCount укаже, якими є справи з наявністю
томів)
            for (int i = 0, e = 0; i < this.n; i++)
            {
                // Індекс рядка з дисперсної матриці, що буде поміщена в матрицю
кодування
                int DRowIdx;

                // Зсув у масиві до елементів i-ой рядка
                int i_n = i * this.n;

                // Якщо основний том відсутній, формуємо рядок матриці
Вандермонда
                if (allVolCount[i] == 0)
                {
                    // Обчислюємо номер рядка матриці Вандермонда, яку потрібно
вставити
                    // на місце даного рядка формованої матриці "FLog"
                    DRowIdx = eccVolToFix[e++];

                    // Указуємо, що даний рядок матриці "FLog" не тривіальна
                    this.FLogRowIsTrivial[i] = false;

                } else
                {
                    // Формуємо в матриці "FLog" нульовий рядок з одиницею на
головній діагоналі
                    // (відповідає наявному основному той)
                    DRowIdx = i;

                    // Указуємо, що даний рядок матриці "FLog" тривіальна
                    this.FLogRowIsTrivial[i] = true;
                }

                // Залежно від типу декодера беремо дані з відповідного масиву
                // (у ньому втримуються як рядки матриці Вандермонда, так і
"тривіальні" рядки,
                //  утримуючі нулі і єдиний елемент "1" на головній діагоналі)
                if (this.eCode_Rid_Solomon_Type ==
(int)Code_Rid_Solomon_Type.Dispersal)
                {
                    int bs = DRowIdx * this.n;

                    // Формування рядка в матриці кодування
                    // ("тривіальні" рядки вже втримуються в матриці "D", вони
вийшли
```

```
                    // "автоматично" на попередньому етапі обробки
MakeDispersal())
                    for (int j = 0; j < this.n; j++)
                    {
                        this.FLog[i_n + j] = this.D[bs + j];
                    }

                } else
                {
                    // Якщо це потрібно - формуємо "тривіальну" рядок...
                    if (this.FLogRowIsTrivial[i])
                    {
                        for (int j = 0; j < this.n; j++)
                        {
                            this.FLog[i_n + j] = 0;
                        }

                        this.FLog[i_n + i] = 1;

                    } else
                    {
                        int bs = (DRowIdx - this.n) * this.n;

                        //...а, інакше, беремо рядок матриці Вандермонда
                        for (int j = 0; j < this.n; j++)
                        {
                            this.FLog[i_n + j] = this.A[bs + j];
                        }
                    }
                }

                // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                // буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    //...указуємо на помилку конфігурації
                    this.configIsOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }
            }

            // Знаходимо зворотну матрицю для "FLog"
            if (!FInv())
            {
                // Указуємо, що кодер зконфігуровано некоректно
                this.configIsOK = false;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return;
            }

            // Обчислюємо логарифми елементів інвертованої матриці
            LogFCalc();
```

```
            // Якщо є передплата на делегата завершення...
            if (OnCode_Rid_Solomon_MatrixFormingFinish != null)
            {
                //...повідомляємо, що екземпляр класу готовий до роботи
                OnCode_Rid_Solomon_MatrixFormingFinish();
            }

            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;

            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }

        #endregion Private Operations

        #region Public Properties

        /// <summary>
        /// Список порядкових номерів наявних томів
        /// </summary>
        public int[] VolList
        {
            get
            {
                if (!InProcessing)
                {
                    return this.volList;

                } else
                {
                    return null;
                }
            }
        }

        #endregion Public Properties
    }
}
```

Файл About.cs - вікно довідки про програму

```
namespace RecoveryData
{
    partial class AboutForm
    {
        /// <summary>
        /// Необхідні змінні розробника.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true якщо керуючі ресурси повинні бути
розташовані, у іншому випадку  false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Необхідний метод для підтримки розробника - не модифікується
        /// зміст цього метода використовується редактором коду.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.developersListBoxdevelopersListBox = new
System.Windows.Forms.ListBox();
            this.Code_Rid_Solomon_IconTimer = new
System.Windows.Forms.Timer(this.components);
            this.okButton_Windows_8 = new PinkieControls.Button_Windows_8();
            this.SuspendLayout();
            //
            // developersListBoxdevelopersListBox
            //
            this.developersListBoxdevelopersListBox.BackColor =
System.Drawing.SystemColors.Control;
            this.developersListBoxdevelopersListBox.BorderStyle =
System.Windows.Forms.BorderStyle.None;
            this.developersListBoxdevelopersListBox.FormattingEnabled = true;
            this.developersListBoxdevelopersListBox.Items.AddRange(new object[]
{
            "МАГІСТЕРСЬКА РОБОТА",
            "",
            "На тему:",
            "",
            " Дослідження та програмна реалізація системи оптимізації
продуктивності та надійності мережних систем зберігання Big Data ",
            "",
            "",
            "Керівник: Коваленко А.С. ",
            "",
            "Розробив: студент Гречушкін Сергій Володимирович",
            "                  гр. КН-23М",
            "",
            "м. Кропивницький 2024"});
            this.developersListBoxdevelopersListBox.Location = new
System.Drawing.Point(11, 6);
            this.developersListBoxdevelopersListBox.Name =
"developersListBoxdevelopersListBox";
```

```
            this.developersListBoxdevelopersListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.developersListBoxdevelopersListBox.Size = new
System.Drawing.Size(330, 182);
            this.developersListBoxdevelopersListBox.TabIndex = 0;
            this.developersListBoxdevelopersListBox.TabStop = false;
            this.developersListBoxdevelopersListBox.SelectedIndexChanged += new
System.EventHandler(this.developersListBoxdevelopersListBox_SelectedIndexChanged
);
            //
            // Code_Rid_Solomon_IconTimer
            //
            this.Code_Rid_Solomon_IconTimer.Interval = 40;
            this.Code_Rid_Solomon_IconTimer.Tick += new
System.EventHandler(this.Code_Rid_Solomon_IconTimer_Tick);
            //
            // okButton_Windows_8
            //
            this.okButton_Windows_8.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.okButton_Windows_8.DefaultScheme = true;
            this.okButton_Windows_8.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.okButton_Windows_8.Hint = "";
            this.okButton_Windows_8.Location = new System.Drawing.Point(266,
177);
            this.okButton_Windows_8.Name = "okButton_Windows_8";
            this.okButton_Windows_8.Scheme =
PinkieControls.Button_Windows_8.Schemes.Blue;
            this.okButton_Windows_8.Size = new System.Drawing.Size(75, 23);
            this.okButton_Windows_8.TabIndex = 0;
            this.okButton_Windows_8.Text = "OK";
            this.okButton_Windows_8.Click += new
System.EventHandler(this.okButton_Windows_8_Click);
            //
            // AboutForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(350, 212);
            this.Controls.Add(this.okButton_Windows_8);
            this.Controls.Add(this.developersListBoxdevelopersListBox);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "AboutForm";
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "Про програму...";
            this.Load += new System.EventHandler(this.AboutForm_Load);
            this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.AboutForm_FormClosing);
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.ListBox developersListBoxdevelopersListBox;
        private System.Windows.Forms.Timer Code_Rid_Solomon_IconTimer;
        private PinkieControls.Button_Windows_8 okButton_Windows_8;

    }
}
```