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
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи передачі та перевірки на цілісність даних у мережі”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-1
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Коростіленко О.М.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук
_____ Лисенко І.А.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

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

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

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

Коростіленку Олегу Максимовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи передачі та перевірки на цілісність даних у мережі*

2. Керівник роботи *Лисенко Ірина Анатоліївна, канд. техн. наук*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 46-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи передачі та перевірки на цілісність даних у мережі*

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

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

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

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

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Лисенко І.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Коростіленко О.М.
(прізвище та ініціали)

АНОТАЦІЯ

Коростіленко О.М. Програмне забезпечення системи передачі та перевірки на цілісність даних у мережі. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи передачі та перевірки на цілісність даних у мережі.

Метою розробки є програмне забезпечення системи передачі та перевірки на цілісність даних у мережі.

Результат роботи – програмна реалізація системи передачі та перевірки на цілісність даних у мережі.

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'ютерна інженерія, перевірка на цілісність даних, комп'ютерна мережа

ABSTRACT

Korostilenko O.M. Software for the system of data transmission and integrity verification in the network. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the system of data transmission and integrity verification in the network.

The purpose of the development is software for the system of data transmission and integrity verification in the network.

The result of the work is the software implementation of the system of data transmission and integrity verification in the network.

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

A convenient user interface has been developed. Instructions for working with software 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 engineering, data integrity checking, computer network

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АБГШ	–	аддітивний білий гаусів шум
БЧХ	–	код Боуза-Чоудхури-Хоквингема
ЕОМ	–	електронно-обчислювальна машина
ІМС	–	інтегральна мікросхема
ARQ	–	автоматичний запит повторної передачі
CD-DA	–	Compact Disc Digital Audio
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. При передачі інформації в сучасних мережах обміну даними застосовується як режим виявлення, так і режим виправлення помилок. Це дозволяє передавати та перевіряти на цілісність дані у мережі У більшості випадків пакети передаються через проміжні центри комутації повідомлень, на яких також можлива їхня перевірка.

Для дослідження методів керування обміном даними в мережах обміну даними необхідний аналіз процесу передачі пакетів у зазначених режимах з урахуванням впливу проміжних центрів комутації повідомлень. При цьому можливі наступні варіанти:

1. Застосування перешкодостійкого коду в режимі виявлення помилок: з перевіркою пакетів одержувачем; з перевіркою на кожному із проміжних центрів комутації повідомлень.

2. Застосування перешкодостійкого кодування в режимі виправлення помилок: з виправленням пакетів одержувачем; з виправленням на кожному із проміжних центрів комутації повідомлень.

Аналіз досліджень і публікацій показав, що застосовуваний метод керування обміном даними повинен забезпечувати найменший відносний середній час доставки інформації й припустиме значення еквівалентної ймовірності помилки. Із практики експлуатації в мережах обміну даними відомо, що ймовірносно-часові характеристики процесу передачі даних залежать від багатьох факторів, які необхідно враховувати при розробці математичних моделей.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи передачі та перевірки на цілісність даних у мережі.

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

– Огляд існуючих систем передачі та перевірки на цілісність даних у мережі.

– Дослідження системи передачі та перевірки на цілісність даних у мережі.

– Програмна реалізація системи передачі та перевірки на цілісність даних у мережі.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі передачі та перевірки на цілісність даних у мережі.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи передачі та перевірки на цілісність даних у мережі, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Цілісність даних є надзвичайно важливою в випробуваннях для забезпечення надійності та достовірності результатів досліджень. У цій роботі представлено вичерпні рекомендації із впровадження практик цілісності даних як коду. Інтегруючи принципи цілісності даних у кодову базу випробування, дослідники можуть автоматизувати перевірки, валідацію та моніторинг, тим самим покращуючи якість даних, дотримання протоколу та загальну ефективність випробувань. Робота охоплює такі ключові аспекти, як розробка перевірки цілісності даних, використання технологій, встановлення моніторингу в реальному часі, забезпечення заходів безпеки та впровадження механізмів контролю якості. Завдяки цьому підходу дослідники можуть підтримувати цілісність даних, оптимізувати процеси та сприяти отриманню надійних і значущих результатів дослідження.

В даному проекті пропонується, для цієї мети використовувати коди Ріда-Соломона.

1.2 Область застосування

Перевірки цілісності даних – це фундаментальні механізми, які використовуються для забезпечення точності та узгодженості даних під час зберігання або передачі. Ці перевірки мають вирішальне значення для виявлення випадкових змін даних, як правило, за допомогою контрольних сум і циклічних надлишкових перевірок (CRC). Обидва методи відіграють значну роль у системах передачі даних, зберігання файлів і виявлення помилок.

Роль контрольних сум:

– Контрольні суми забезпечують простий метод перевірки цілісності даних шляхом підсумовування значень набору даних. Потім ця сума зберігається як значення контрольної суми, яке можна перерахувати та порівняти з оригіналом для перевірки цілісності.

– Хоча базові контрольні суми є простими, вони мають обмеження у виявленні певних типів помилок, наприклад змін, які скасовують одна одну.

Перевірка циклічної надлишковості (CRC):

– CRC – це складніший код виявлення помилок, який використовується спеціально для виявлення випадкових змін необроблених даних. Він використовує поліноміальне ділення для обчислення контрольної суми, відомої як значення CRC.

– CRC зазвичай використовуються в мережевих комунікаціях і зберіганні файлів через їхню ефективність у виявленні пакетних помилок. Вони забезпечують більш високий ступінь надійності порівняно з базовими контрольними сумами.

Важливість перевірки цілісності даних:

– Забезпечення цілісності даних має вирішальне значення для запобігання втраті даних, підвищення стійкості систем до пошкоджень і забезпечення впевненості в тому, що дані залишаються незмінними під час передачі чи зберігання.

– І контрольні суми, і CRC інтегровані в численні програми та системи для перевірки цілісності даних. Це включає протоколи передачі файлів, системи зберігання дисків і мережевий зв’язок.

Виклики та міркування:

– Потрібно знати, що хоча контрольна сума та CRC є ефективними для виявлення помилок, жоден метод не може виправити помилки; вони можуть лише визначити, що сталася помилка.

– Вибір правильного методу передбачає баланс між складністю, споживанням ресурсів і можливістю виявлення помилок. CRC часто потребує більшої потужності обробки, але забезпечує більший захист від помилок.

Розуміння та впровадження перевірок цілісності даних, таких як контрольні суми та CRC, має важливе значення для підтримки цілісності даних у різних системах і програмах. Використовуючи ці методи, організації можуть краще зберегти точність і надійність своїх даних.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи передачі та перевірки на цілісність даних у мережі, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Розглянемо існуюче програмне забезпечення системи передачі та перевірки на цілісність даних у мережі.

MD5 File Checker

При завантаженні файлів часто трапляється, що файл завантажується "битим" за тих або інших причинах. Щоб напевно впевнитися в ідентичності завантаженого файлу на сервері, необхідно порівняти їхні контрольні суми. Для цієї мети можна використовувати алгоритм підрахунку контрольних сум md5. При додаванні файлу на сервер у нього обчислюється ця сама контрольна сума, побачити її можна на сторінці інформації про файл. Щоб обчислити цю контрольну суму вже для завантаженого файлу на вашім комп'ютері, скористайтеся спеціально написаною програмою:

Програма не вимагає установки. Для користувачів операційних систем Linux, Unix, MacOS і інших – аналогічні програми дивитеся прямо в складі своєї ОС.

Коротка інструкція з роботи із програмою MD5 File Checker:

– Запускаємо програму. На сторінці з будь-яким файлом у блоці інформації про файл знаходимо поле з контрольною сумою md5 і копіюємо її в буфер обміну.

– Вставляємо в поле для перевірки №2. Вибираємо файл, що хочемо перевірити в поле №1. Натискаємо кнопку "Перевірити". У результаті одержуємо повідомлення з результатом роботи програми. Наприклад "MD5 сума файлу

збігається з контрольною сумою. Файл не ушкоджений" або "MD5 сума файлу не збігається з контрольною сумою. Можливо, файл ушкоджений". Робимо виводи.

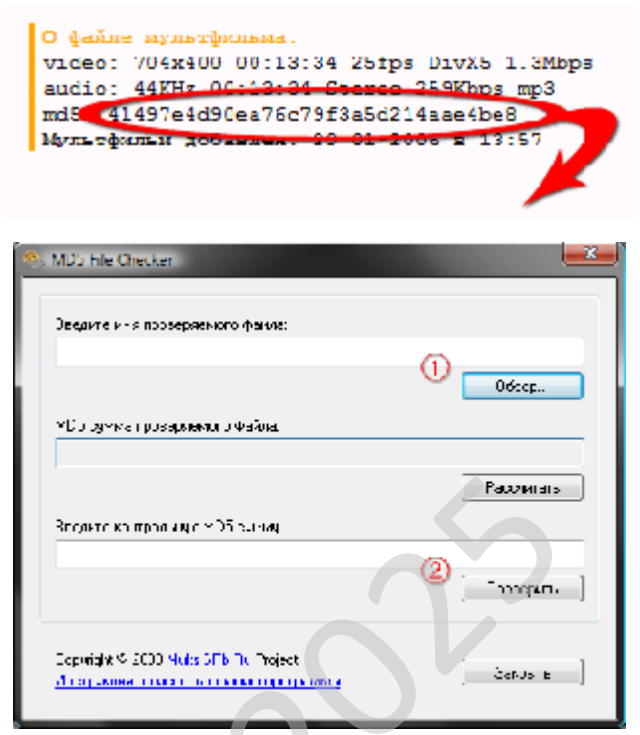

Рисунок 2.1 – Інтерфейс користувача MD5 File Checker

Додаткова можливість програми – розрахунок контрольної суми md5 для довільного файлу і її вивід користувачеві.

EF CheckSum Manager

Вийшла нова версія програми для перевірки цілісності файлів. За допомогою EF CheckSum Manager можна перевіряти контрольні суми або ж створювати нові для окремо взятих файлів або цілих папок. Для перевірки цілісності даних програма використовує файли у форматах SFV, MD5 і SHA1. Такі файли займають зовсім небагато місця, тому можуть бути записані на диски разом з основними даними. Перевірка цілісності може використовуватися не тільки для файлів, які записуються на CD/DVD, але й при пересиланні поштою. Крім цього, EF CheckSum Manager зручно використовувати для перевірки файлів, які були завантажені з Інтернету.

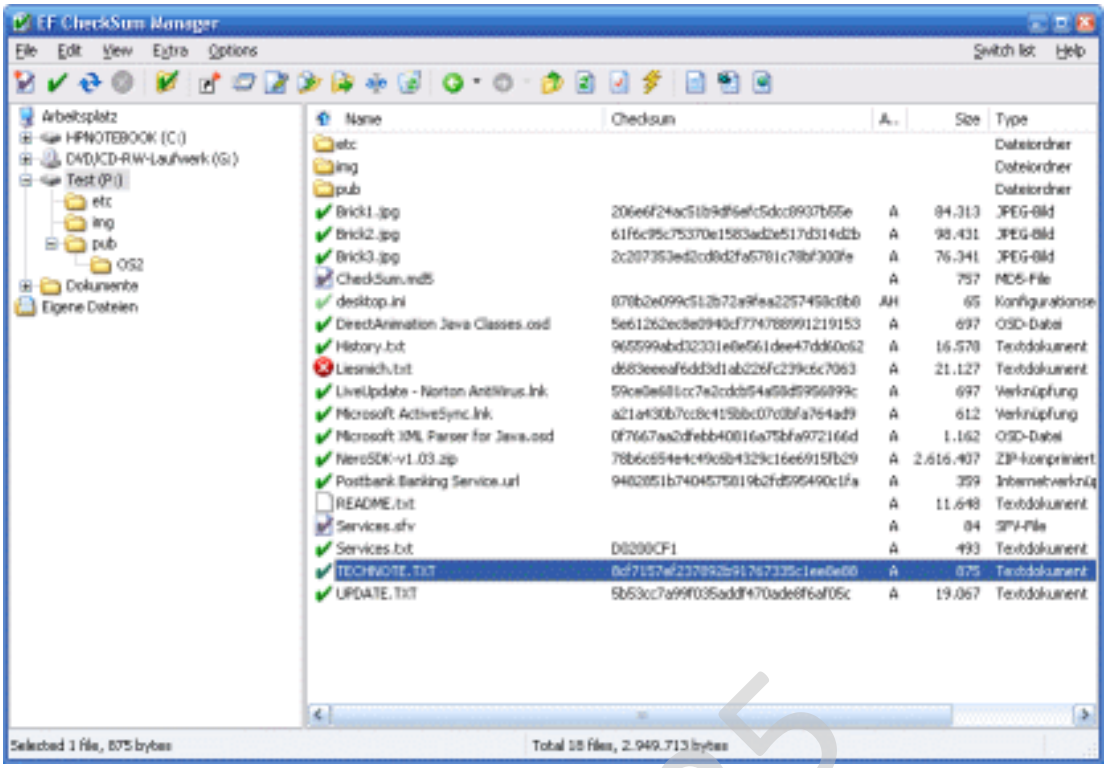

Рисунок 2.2 – Інтерфейс користувача EF CheckSum Manager 6.10

HashTab

HashTab представляє із себе розширення провідника Windows і плагін для Mac Finder для перевірки цілісності й дійсності файлів за допомогою обчислення контрольної суми. HashTab підтримує безліч алгоритмів хешування, таких як CRC, MD5, SHA1, SHA2, SHA3/Keccak, RipeMD і Whirlpool, а так само BitTorrent Info Hash і генерацію Magnet-посилань.

Після установки HashTab, кликніть правою кнопкою миші по будь-якому файлі. В Windows, виберіть «Властивості», і ви побачите нову вкладку «Хеш-суми файлів». В Mac, виберіть «File Hashes». В Mac OS X 10.8 меню «File Hashes» розташовано в підменю «More». Вікно «Хеш-суми файлів» відображає всі хеші для обраного файлу. Ви можете настроїти, які хеші будуть обчислюватися й виводитися на екран. Ви можете хешувати інші файли для порівняння. Ви також можете вставити текст хешу, у такий спосіб вам не прийдеться очами порівнювати MD5 хешу, індикатор HashTab покаже, є чи збігу.

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` – крапку входу додатка – інкапсуюється 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.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи передачі та перевірки на цілісність

даних у мережі.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

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

б) вибрати та обґрунтувати методику побудови системи 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

Наведемо опис функціонування системи. Як було визначено вище, призначенням системи є передача та перевірка на цілісність даних у мережі, за допомогою перешкодостійкого кодування. Високий рівень складності та той факт, що програмне та апаратне забезпечення настільки складно пов’язані, означає, що система може бути дуже чутливою до помилок [1], [2], [3].

Зокрема, помилки викликають велике занепокоєння при розробці систем високої доступності або систем, що використовуються в електронному ворожому середовищі.

Космічні програми, де система не може дозволити собі збій під час польоту, вразливі до програмних помилок.

Системи моніторингу ядерної енергетики, де одиничний збій може спричинити серйозні руйнування, і системи реального часу, де пропущений термін може вважатися помилковою дією та можливим збоєм системи, є кількома іншими прикладами, коли м’які помилки є критичною проблемою [4], [5].

Миттєвий збій у банківських операціях через м’які помилки може спричинити величезну різницю в балансі.

Якщо помилка викликає перевороти 1→0 бітів у старшому біті реєстру, що зберігає суму грошей, депонованих на банківському рахунку, то наслідком може бути несподівана зміна балансу [6].

Пошкоджене проміжне значення може призвести до пошкодження всіх наступних обчислень і виконання всієї програми.

Системи реального часу та вбудовані системи тепер є центральною частиною нашого життя.

Надійне функціонування систем реального часу є головною проблемою для мільйонів користувачів, які щодня залежать від цих систем [7]. Отже, відмовостійкість є важливою для систем реального часу [8]. Для розробки відмовостійкої системи виявлення та виправлення помилок є дуже важливими.

Помилка програмного забезпечення виникає, коли інформація передається від одного вузла до іншого вузла. Виявлення та виправлення помилки саме в цей момент є обов'язковим. Існуючі методи виявлення та виправлення помилок спричиняють високу надмірність інформації та/або високу надмірність інформації. З цієї причини потрібен ефективний підхід до виявлення та виправлення помилок

Багато дослідників уже запропонували кілька методологій, які займаються аналізом та оптимізацією щодо виявлення та виправлення помилок. Коди низької щільності перевірки парності (LDPC) були вперше представлені Галлагером [9]. Звичайний код Галлагера має контрольну матрицю парності з рівномірною вагою стовпця j і рівномірною вагою рядка k [10]. За останні кілька років було розроблено кілька кодів перевірки парності з низькою щільністю (LDPC) з продуктивністю, дуже близькою до обмеження Шеннона [11].

Санкаранараянан і Васік порівняли матрицю парності з кількома методами виявлення та виправлення помилок [2]. Вони описали підхід ортогоналізації перевірки парності. Парність слова перевіряється після читання слова з пам'яті. Слово приймається, якщо зчитані біти парності правильні. Якщо біти парності неправильні, виявляється помилка, але її неможливо виправити. Код з виправленням помилок використовує декілька бітів перевірки парності, які зберігаються разом зі словом даних у пам’яті. Кожен контрольний біт є бітом парності для групи бітів у слові даних. Коли слово зчитується з пам'яті, парність кожної групи, включаючи контрольний біт, оцінюється. Якщо парність правильна для всіх груп, це означає, що жодної виявленої помилки не сталося. Двовимірна парність також може виявити (але не виправити!) будь-яку комбінацію двох помилок у пакеті.

Ібрагім описав код Хеммінга та циклічний код [3]. Ця теорія узагальнена математиком Гокквенгхемом і дає початок сімейству кодів BCH. Коди Хеммінга двійкові відразу здаються кодами BCH. Коди Хеммінга можуть виявляти та виправляти однобітові помилки. Іншими словами, відстань Хеммінга між переданим і отриманим кодовими словами повинна бути рівною або одиниці для надійного зв'язку. Крім того, він може виявити (але не виправити) до двох одночасних бітових помилок. Він поєднав код Хеммінга та циклічний код, щоб ефективно виявляти та виправляти помилки.

Коди циклічної перевірки надлишковості (CRC) – це спеціальна підмножина лінійних блокових кодів, які дуже популярні в цифровому зв’язку. Коди CRC мають властивість циклічного зсуву; коли будь-яке кодове слово повертається вліво або вправо на будь-яку кількість бітових цифр, результуючий рядок все ще залишається словом у кодовому просторі [2]. Паралельна схема CRC одночасно обробляє кілька бітів даних [12]. Теоретичні аспекти кодування циклічних надлишкових кодів (CRC) розглядаються [13]. Шпрахманн описав генерацію схеми CRC для виявлення та виправлення помилок [3]. Ця властивість робить кодування та декодування дуже простим і ефективним у реалізації за допомогою простих регістрів зсуву. Більш складні методи виявлення (і виправлення) помилок використовують властивості скінченних полів і поліномів над такими полями [14]. Циклічна перевірка надлишковості розглядає блок даних як коефіцієнти до полінома, а потім ділить на фіксований, заздалегідь визначений поліном. Коефіцієнти результату ділення приймаються як надлишкові біти даних, CRC. Після отримання можна повторно обчислити CRC з корисних бітів і порівняти його з отриманим CRC. Amismatch вказує на те, що сталася помилка. Основним недоліком використання кодів CRC є те, що він має лише можливості виявлення помилок. Вони не можуть виправити будь-які помилки в даних, виявлених у пункті призначення, і дані повинні бути передані знову, щоб отримати повідомлення. З цієї причини коди CRC зазвичай використовуються в поєднанні з іншим кодом, який забезпечує виправлення помилок. Якщо коди

CRC є єдиними, що використовуються для програми, необроблений BER каналу, як правило, надзвичайно низький, а дані не є критичними щодо часу. Вони підходять для магнітних і оптичних накопичувачів, де можливий простий запит на повторну передачу для виправлення бітових помилок. Худак та ін. порівняли кілька методів відмовостійкого програмного забезпечення, включаючи відмовостійкість на основі алгоритму на основі частоти помилок [15]. Техніки були класифіковані на основі надійності та вартості. Тут він описав різні типи схем виявлення та виправлення помилок.

Бодрі визначив обчислювальну потужність як показник продуктивності для вивчення взаємодії між продуктивністю та надійністю [16]. Автор включив обчислювальну потужність у моделювання надійності резервованих систем. Автор визначив обчислювальну потужність як кількість корисних обчислень на одиницю часу, доступну в системі. Оскільки автор оцінював системи, що поступово деградують, автор визначив обчислювальну потужність як функцію стану системи. Для збільшення обчислювальної потужності система потребує якомога перших схем виявлення та виправлення помилок. Запропонований метод є швидким і має властивість автоматичного виявлення та виправлення помилок.

3.2 Розробка структурної схеми

Структурна схема системи наведена на рисунку 3.1. На ньому наведено комп'ютерну мережу, у якій дані, які передаються, перевіряються на цілісність за допомогою кодів Ріда-Соломона.

Мережа складається з наступних елементів.

- Gw – програмний маршрутизатор під керуванням FreeBSD.
- Edge – Active Directory Domain Controller.
- Onyx – production сервер під керуванням FreeBSD.
- Lemur – сервер під керуванням FreeBSD.
- Vectra – сервер під FreeBSD.

– Hydra – файл-сервер.

Активне устаткування:

– 2950g-ei – комутатор Cisco Catalyst 2950G-EI. .

– sw-416-3, sw-416-1, sw-229 – керуємі комутатори доступу Intel Express 460T.

– sw-417— некерований комутатор доступу Hewlett Packard HP2524.

– sw-422 керуємий комутатор Intel Express.

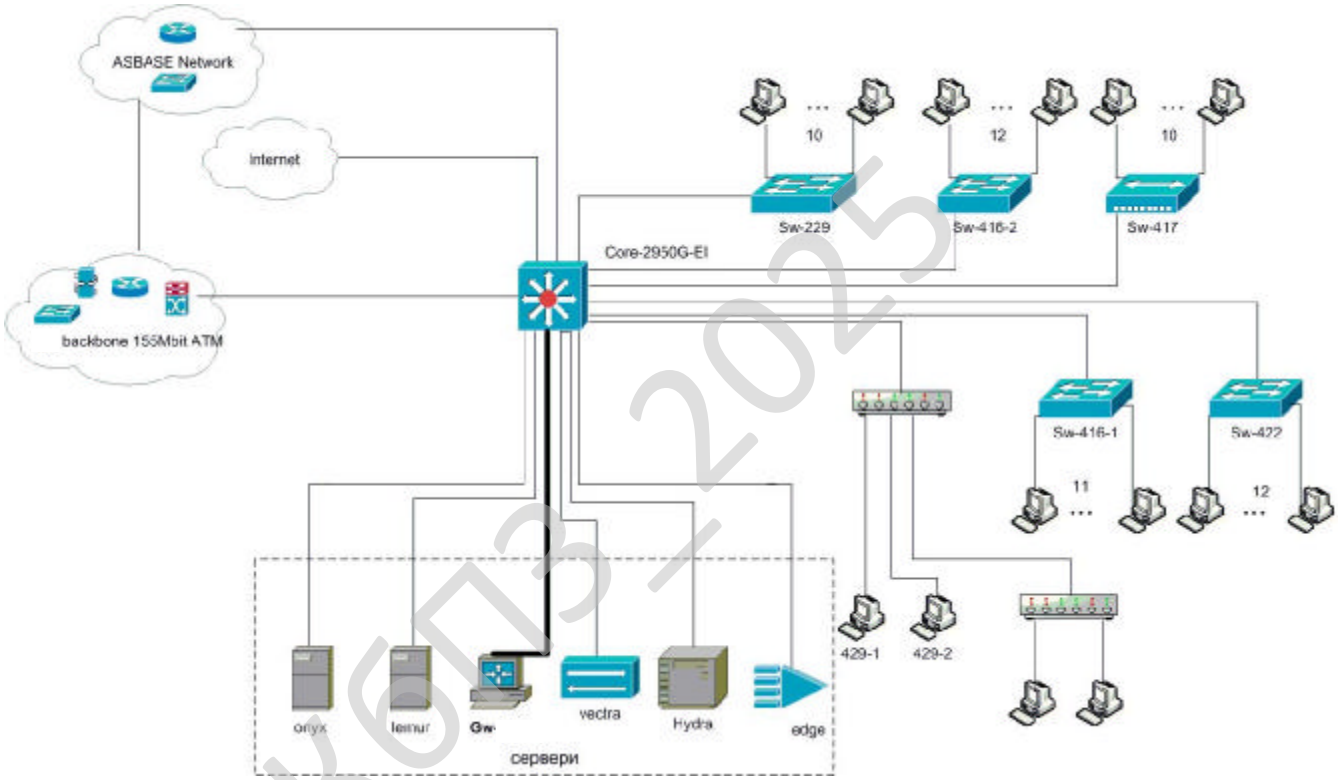

Рисунок 3.1 – Структурна схема системи

Клієнтські місця повинні працювати під керуванням ОС Windows, мати веб-браузер.

Перевірки цілісності даних, такі як контрольні суми та CRC (перевірки циклічної надлишковості), є життєво важливими для забезпечення надійності даних, що зберігаються або передаються у вбудованому програмному

забезпеченні. Вони допомагають виявити випадкове пошкодження даних і є важливою частиною стратегій виявлення помилок.

Навіщо використовувати контрольні суми та CRC:

– Виявлення помилок: вони в основному використовуються для виявлення помилок у даних, які могли виникнути під час зберігання або передачі.

– Ефективність: як контрольні суми, так і CRC ефективні з точки зору обчислень і забезпечують хороший баланс між можливістю виявлення помилок і швидкістю.

Вибір правильного алгоритму

Вибираючи алгоритм для перевірки цілісності даних у програмі, враховуйте такі фактори, як розмір даних, необхідна швидкість і прийнятна частота виявлення помилок. Загальні алгоритми включають:

– Прості контрольні суми: підходить для невеликих блоків даних і низьких вимог до безпеки.

– CRC32: ідеально підходить для виявлення помилок під час передачі великих даних, широко використовується в мережевих системах.

Інтеграція перевірок у процес програми

Щоб інтегрувати ці перевірки у програму, виконайте такі дії:

– Підготовка даних: визначте блоки даних, до яких буде застосовано перевірку цілісності, наприклад дані конфігурації, зображення програм або мережеві повідомлення.

– Кодування: зберігайте або передайте контрольну суму або значення CRC поряд з вихідними даними.

– Перевірка: після отримання або отримання даних повторно обчисліть контрольну суму або CRC і порівняйте її з переданим значенням, щоб перевірити цілісність даних.

Тестування та валідація

Тестування має вирішальне значення. Переконайтеся, що ваше програмне забезпечення витончено виконує як виявлення, так і виправлення помилок.

– Введіть відомі помилки в тестові дані, щоб ваша система виявила їх.

– Переконайтеся, що перевірки істотно не впливають на продуктивність програми.

Застосування цих методів може значно підвищити надійність і стійкість вашого програмного забезпечення, зменшивши ризик невиявленого пошкодження даних.

3.3 Розробка функціональної схеми

Формальний опис

Поліном, що породжує

Визначення 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 $g_r = 1$.

Матриця, що породжує

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

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

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

Матрицю *G* можна записати в символьній формі:

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

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:

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

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 $GF(q)$, коренями якого є $d - 1$ підряд, що йдуть ступенів, $\alpha^{l_0}, \alpha^{l_0+1}, \dots, \alpha^{l_0+d-1}$ 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 $2\underline{t}$ 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 $l + (m - 1) <= mt$.

Застосування

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

Запис і зберігання інформації

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.

Запис на CD/DVD-ROM

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.

Бездротовий і мобільний зв'язок

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.

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.

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Кодування кодеком Ріда-Соломона

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.

Декодування Ріда-Соломона

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 $2t$, 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 нацело, 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 кореня

На цьому етапі шукаються корені 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 – корені знайдені.

Визначення характеру помилки і її виправлення

По синдрому помилки й знайдених корінь полінома за допомогою алгоритму Форні 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.

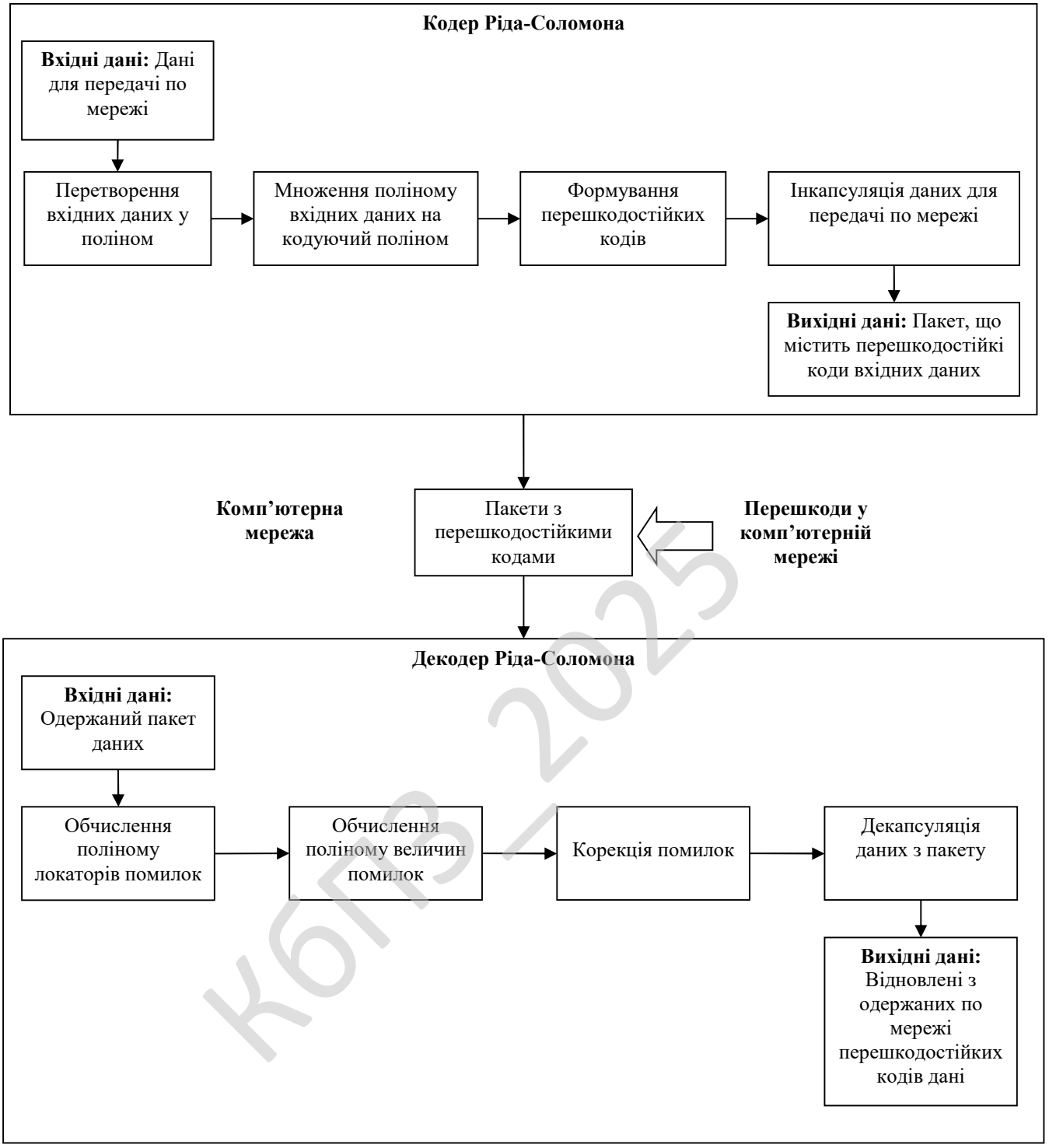

Рисунок 3.2 – Функціональна схема системи

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

– дані, які потрібно захистити;

– поліном для кодування;

– перешкодостійки коди.

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

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. Після початку роботи розробленого ПЗ ми потрапляємо до головного блоку системи звідки через ланку дій відбувається наступне:

– Виведення інтерфейсу ПЗ.

– Підсистема налаштування.

– Параметри журналювання.

– Встановлення кількості дублюючих IP-пакетів.

– Параметри шифрування.

– Підсистема кодування.

– Створення IP-пакетів.

– Шифрування.

– Підсистема декодування.

– Перевірка цілісності даних.

– Відновлення пошкоджених даних.

– Дешифрування.

– Введення ключа шифрування.

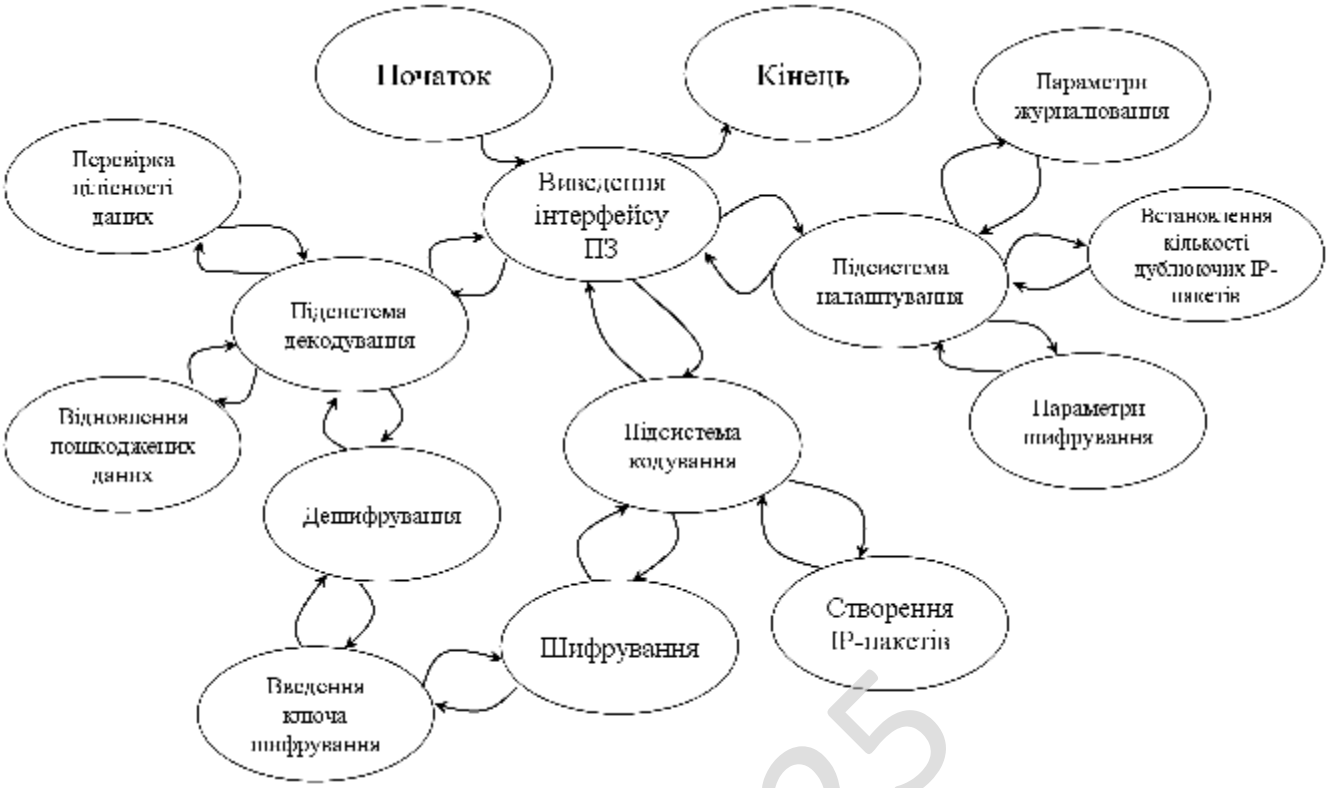

Рисунок 3.3 – Діаграма взаємодії процесів

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

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем.

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

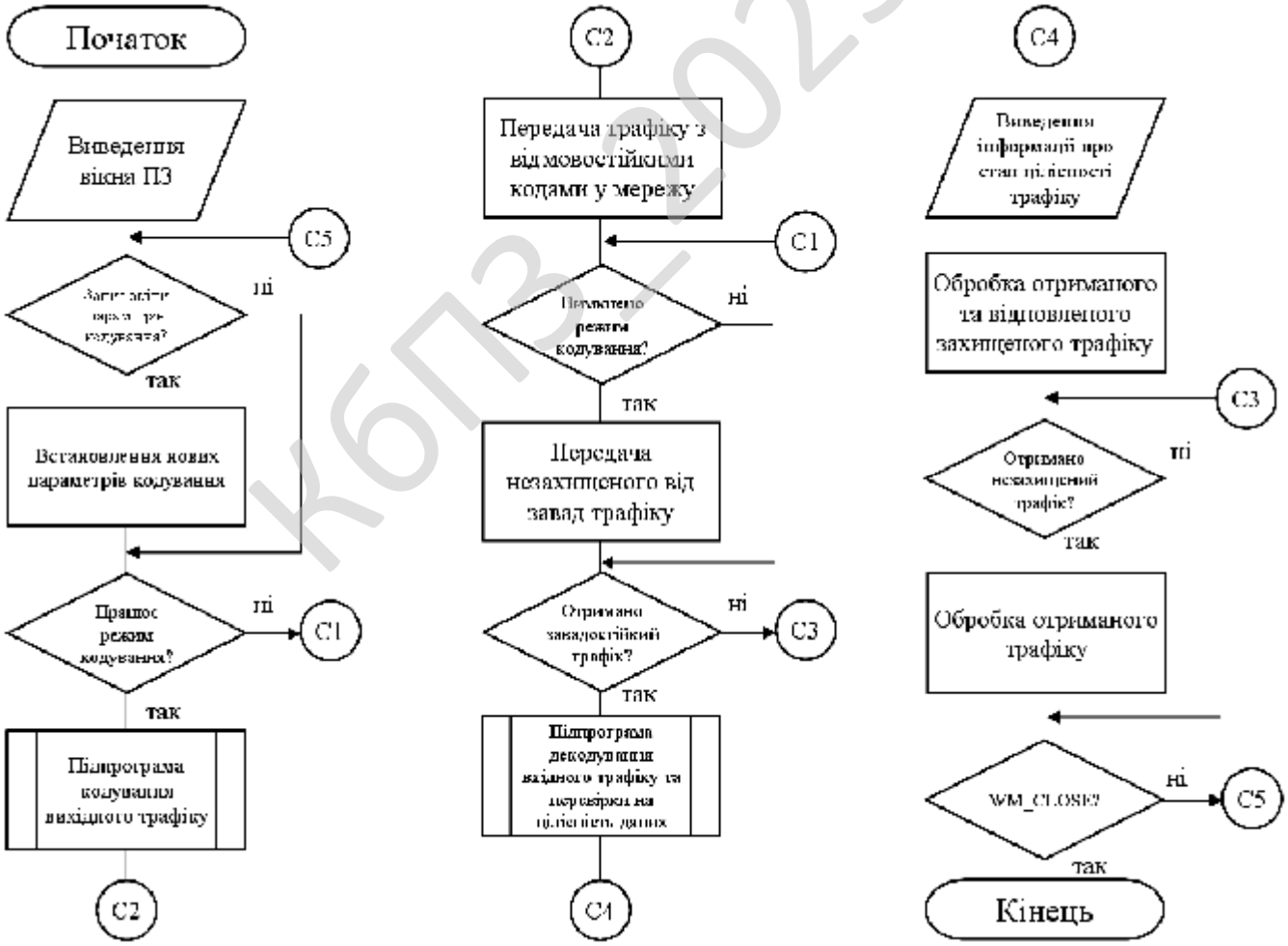

Рисунок 4.1 – Блок схема основної програми

З якої 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.

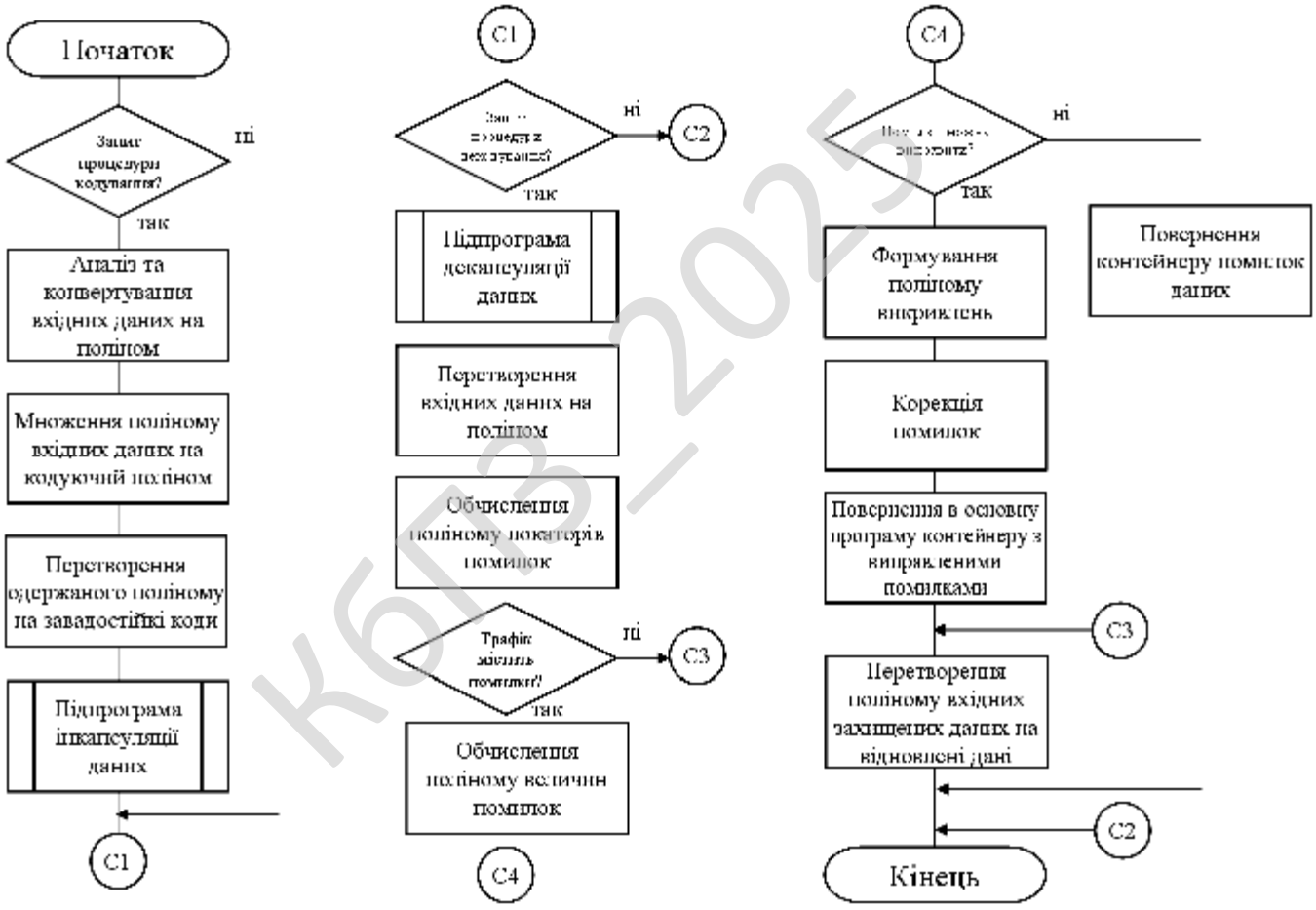

Рисунок 4.2 – Блок схема підпрограми

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 я враховував, 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 передачі та перевірки на цілісність даних у мережі.

При складанні блок-схем програмного забезпечення і напрацювання алгоритмів я зіткнувся з масою проблем, які вимагали напрацювання процедур і функцій над основною проблематикою. Для чого були створені додаткові класи, типи даних і константи, що забезпечило вирішення проблем.

Опис алгоритмів функціонування системи

Розглянемо кодування алгоритмом Ріда-Соломона який реалізується за допомогою класу RS_Raid_Encoder. Базовий конструктор кодера, в якому створюється об’єкт GF16 для роботи з елементами поля Галуа:

```
public RS_Raid_Encoder(int dataCount, int eccCount)
        {
            SetConfig(dataCount, eccCount, (int)RSType.Alternative);
            this.eGF16 = new GF16();
        }
```

Для множення матриці кодування на вхідний прологарифмований вектор використовується метод Process:

```
public bool Process(int[] dataLog, ref int[] ecc)
        {
// Якщо кодер зконфігуровано некоректно, обробка неможлива!
            if (!this.configIsOK)
            {
                return false;
            }
            int[] GF16Exp = this.eGF16.GFExpTable;
// Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.m; i++)
            {
                int mulSum = 0;
                int i_n = i * this.n;
                for (int j = 0; j < this.n; j++)
                {
                    mulSum ^= GF16Exp[this.FLog[i_n + j] + dataLog[j]];
```

```
            }
            ecc[i] = mulSum;
        }
        return true;
    }
```

Для скорочення часу обрахунку копіюємо покажчик на масив експонент eGF16.GFExpTable. Таким отримується GF16Exp.

```
int[] GF16Exp = this.eGF16.GFExpTable;
```

Заповнення матриці кодування даними відбувається з використанням процедури FillFLog:

```
protected override void FillFLog()
    {
        if (this.mainConfigChanged)
        {
            if (this.eRSType == (int)RSType.Dispersal)
            {
                if (!MakeDispersalMatrix())
                {
                    this.configIsOK = false;
                    this.finished = true;
                    this.finishedEvent[0].Set();
                    return;
                }
            } else
            {

                if (!MakeAlternativeMatrix())
                {
                    this.configIsOK = false;
                    this.finished = true;
                    this.finishedEvent[0].Set();
                    return;
                }
            }
            this.FLog = new int[this.m * this.n];
            for (int i = 0; i < this.m; i++)
            {
                int i_n = i * this.n;
                if (this.eRSType == (int)RSType.Dispersal)
                {
```

```
                        for (int j = 0; j < this.n; j++)
                        {
this.FLog[i_n + j] = this.eGF16.Log(this.D[((this.n + i) * this.n) + j]);
                        }
                    } else
                    {
                        for (int j = 0; j < this.n; j++)
                        {
                            int idx = i_n + j;
                            this.FLog[idx] = this.eGF16.Log(this.A[idx]);
                        }
}
```

Якщо основна конфігурація змінилася виконується формування дисперсної матриці "D", інакше відбувається формування альтернативного заповнення матриці "A".

Перейдемо до розгляду методів декодування.

Декодування алгоритмом Ріда-Соломона реалізується за допомогою класу RSRaidDecoder.

Заповнення матриці "FLog" (матриці декодера) даними відбувається із використанням процедури FillFLog:

```
protected override void FillFLog()
```

Якщо довжина вектора наявних томів менше кількості, необхідних для відновлення, вказуємо на помилку конфігурації, активуємо індикатор актуального стану змінних-членів та встановлюємо подію завершення обробки:

```
if (this.volList.Length < this.n)
                {
                    this.configIsOK = false;
                    this.finished = true;
                    this.finishedEvent[0].Set();
                    return;
                }
```

Для перевірки томів використовуємо наступні лічильники:

– Вектор лічильників всіх томів:

```
int[] allVolCount = new int[this.n + this.m]
```

– Лічильник кількості стертих основних томів:

```
int dataVolMissCount = this.n
```

Далі ініціалізується масив лічильників всіх томів:

```
for (int i = 0; i < (this.n + this.m); i++)
            {
                allVolCount[i] = 0;
            }
```

Створюється вектор ECC-томів для заповнення пробілів, створених загубленими основними томами:

```
int[] eccVolToFix = new int[this.m];
```

Далі проводимо аналіз складу представлених томів на предмет наявності основних томів. Обчислюємо номер поточного тому i, якщо номер тому відповідає припустимому діапазону, збільшуємо лічильник всіх томів allVolCount.

Якщо том є основним, фіксуємо цей факт, зменшуючи dataVolMissCount. Якщо номер тому виходить за межі припустимого діапазону, вказуємо на помилку конфігурації, активуємо індикатор актуального стану змінних-членів та встановлюємо подію завершення обробки:

```
for (int i = 0; i < this.n; i++)
            {
                int currVol = Math.Abs(this.volList[i]);
                if (currVol < (this.n + this.m))
                {
                    ++allVolCount[currVol];
                    if (currVol < this.n)
                    {
                        --idataVolMissCount;
                    }

                } else
                {
                    this.configIsOK = false;
                    this.finished = true;
                    this.finishedEvent[0].Set();
                    return;
                }
}
```

Наступним кроком перевіряємо лічильники томів на помилкове дублювання:

```
for (int i = 0; i < (this.n + this.m); i++)
      {
          if (allVolCount[i] > 1)
          {
              this.configIsOK = false;
              this.finished = true;
              this.finishedEvent[0].Set();
              return;
          }
      }
```

Якщо деякий том був зазначений більш ніж один раз вказуємо на помилку конфігурації, активуємо індикатор актуального стану змінних-членів та встановлюємо подію завершення обробки.

Якщо перевірка на несуперечність не виявила проблем, починаємо формувати матрицю FLog.

Для кожного загубленого основного тому шукаємо том для відновлення. Рухаємося за списком томів доти, поки не знайдемо том для відновлення для заповнення "дірки" (основні томи мають номери менше this.n (при нумерації з нуля)). :

```
for (int i = 0, j = 0; i < dataVolMissCount; i++)
            {
                while (this.volList[j] < this.n)
                {
                    j++;
                }
                eccVolToFix[i] = this.volList[j];
                j++;
}
```

Для заміни загубленого основного тому зберігаємо номер тому в eccVolToFix. Далі проходимо рядки матриці, перевіряючи за допомогою allVolCount відсутність ушкоджень. Такому сценарію відповідає розташуванню одиниць на головній діагоналі.

Якщо основний том відсутній, формуємо рядок матриці Вандермонда. Для цього обчислюємо номер рядка матриці Вандермонда, яку потрібно вставити на місце даного рядка формованої матриці FLog та вказуємо, що даний рядок

матриці FLog не тривіальний. Якщо пропусків немає, формуємо в матриці FLog нульовий рядок з одиницею на головній діагоналі (відповідає наявному основному тому) та вказуємо, що даний рядок матриці FLog тривіальний.

```
for (int i = 0, e = 0; i < this.n; i++)
            {
                int DRowIdx;

                int i_n = i * this.n;
                if (allVolCount[i] == 0)
                {
                    DRowIdx = eccVolToFix[e++];

                    //                    this.FLogRowIsTrivial[i] = false;
                } else
                {
                    DRowIdx = i;
                    this.FLogRowIsTrivial[i] = true;
                }
```

Залежно від типу декодера беремо дані з відповідного масиву (у ньому містяться як рядки матриці Вандермонда, так і тривіальні рядки, що містять нулі і єдиний елемент "1" на головній діагоналі)

Формування рядка в матриці кодування (тривіальні рядки вже втримуються в матриці "D", вони вийшли автоматично на попередньому етапі обробки MakeDispersal()):

```
if (this.eRSType == (int)RSType.Dispersal)
                {
                    int bs = DRowIdx * this.n;

                    for (int j = 0; j < this.n; j++)
                    {
                        this.FLog[i_n + j] = this.D[bs + j];
                    }
                } else
                {
                    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;
                for (int j = 0; j < this.n; j++)
                {
                    this.FLog[i_n + j] = this.A[bs + j];
                }
            }
}
```

Для пошуку матриці, оберненої до FLog, використовується метод 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 означає збій кодека, тому ситуація з виявленням нуля сприймається винятково як помилка). Розглянемо роботу процедури FInv, яка повертає true, якщо матриця FLog має:

```
private bool FInv()
{
double allStageIter   = this.iterOfFirstStage + this.iterOfSecondStage;
int percOfFirstStage  = (int)((100.0 * this.iterOfFirstStage)  /
allStageIter);
int percOfSecondStage = (int)((100.0 * this.iterOfSecondStage) /
allStageIter);
// Цикл вибору елемента "pivot"
            for (int k = 0; k < this.n; ++k)
            {
// Якщо даний рядок тривіальний – просто переходимо на нову ітерацію
                if (this.FLogRowIsTrivial[k])
                {
                    continue;
                }
// Зсув у масиві до елементів k-ого рядку
                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 зворотної
                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]);
                    }
                }
// Ділення 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);
                    }
                }

            return true;
        }
```

4.2 Захист розробленого програмного забезпечення

Дані які використовуються у даній роботі захищаються алгоритмом ДСТУ 7624:2014 («Калина»). При розробці національного стандарту ДСТУ 7624:2014 (блоковий шифр «Калина» і режими його роботи) було ухвалене рішення забезпечити прозорість проектування й використовувати консервативний підхід із застосуванням добре досліджених конструкцій, що забезпечують запас стійкості для безпечного застосування алгоритму в умовах істотного прогресу криптоаналітичних технік і засобів обробки даних.

Національний стандарт підтримує розмір блоку й довжину ключа шифрування 128, 256 і 512 біт (довжина ключа дорівнює розміру блоку або у два рази перевищує його), забезпечуючи нормальний, високий і надвисокий рівень стійкості (зараз це єдиний у світі стандарт блокового шифрування, що підтримує 512-бітові симетричні ключі). Різні варіанти забезпечують гнучкість вибору

параметрів для розроблювачів систем криптографічного захисту, що дозволяє одержати як найвищий рівень швидкодії, так і найбільший запас стійкості перетворення.

Високорівнева конструкція використовує добре досліджену Square-подібну SPN-структуру, застосовувану в алгоритмах AES/Rijndael, Whirlpool, «Стрибог», «Коник» і багатьох інших. Циклове перетворення побудоване на базі таблиць підстановки (S-блоків) і множення на МДР-матрицю над кінцевим полем, забезпечуючи необхідні криптографічні властивості. Застосування саме такої конструкції дозволяє забезпечити доказову стійкість до диференціального, лінійному й ін. видам криптоаналізу, одночасно забезпечуючи ефективну реалізацію на широкому спектрі програмних і програмно-апаратних платформ. При виборі розміру МДР-матриці був прийнятий в увагу розмір кешу L1 сучасних і перспективних процесорів, що дозволило оптимізувати швидкодія програмної реалізації шифру.

Альтернативний варіант, який розглядався при розробці циклової функції, – ARX перетворення (Addition-rotation-xor). Цей підхід реалізований у шифрі SPECK (розроблений Агентством національної безпеки США й переданий у відкритий доступ), у блокових алгоритмах, на основі яких побудовано сімейство геш-функцій SHA-0,1,2 і ін. Перевагою походу є компактність і швидкодія перетворення. Але, у той же час, є й істотний недолік, пов'язаний з відсутністю методів, що дозволяють виконати строге аналітичне обґрунтування криптографічної стійкості таких розв'язків. Навіть із наймогутнішими у світі можливостями для аналізу, США кілька раз були змушено модифікувати свої стандарти гешування через знайдені уразливості: з 1993 по 1995 рр. діяв SHA-0, з 1995 по 2001 рр. застосовувався SHA-1, з тих пор використовується SHA-2. Через питання, що піднімаються, до стійкості й цієї версії, у США з 2008 по 2012 рр. був проведений міжнародний конкурс SHA-3. До теперішнього часу розроблений проект стандарту FIPS 202, що описує нову криптографічну геш-функцію.

Таким чином, консервативний і прозорий підхід до проектування нового національного стандарту України обумовив вибір добре перевіреної конструкції на базі S-блоків і лінійного перетворення, для якої можливо забезпечити доказову стійкість до різних видів криптоаналізу.

При порівнянні характеристик S-блоків і інших симетричних перетворень, можна відзначити, що саме національний стандарт України забезпечує найбільшу нелінійність булевих функцій, що дає додатковий запас стійкості до лінійного криптоаналізу. Ще більше значення нелінійності для взаємо- однозначної підстановки можна забезпечити, застосовуючи, наприклад, аффінно-тквівалентні статечні функції в кінцевому полі, але такі перетворення, використані в AES, Camellia і ін. алгоритмах, ставлять шифр під погрозу реалізації алгебраїчної атаки (цей метод криптоаналізу був успішно застосований проти шифру Keeloq, використовуваного в системах автомобільної безпеки).

У якості схеми розгортання ключів була запропонована нова конструкція з наступними властивостями:

– забезпечення криптографічної стійкості до відомих методів аналізу, відсутність «слабких» ключів, які можуть погіршити властивості перетворення;

– зручність програмної й програмно-апаратної реалізації (для формування циклових ключів застосовуються тільки операції, використовувані при шифруванні);

– висока обчислювальна складність відновлення ключа шифрування по одному або декільком цикловим ключам.

Остання властивість забезпечує додатковий захист до атак на реалізацію, коли зловмисник намагається атакувати інженерні розв'язки (вимірюючи споживаний пристроєм струм, навмисне викликаючи збої в роботі через навмисний перегрів шифратора та ін.). Ця особливість є істотною перевагою для ряду додатків, зокрема, при реалізації шифрування на смарт-картах, USB-токенах та ін., коли ключ прошитий у пристрої й повинен бути захищений від

зовнішнього доступу (наприклад, у модулях доступу до платних цифрових ТБ-каналівм і ін.)

Кількість циклів шифрування залежить від довжини ключа: 10 циклів для 128-бітового, 14 циклів для 256-бітового й 18 циклів для 512-бітового ключа шифрування.

У порівнянні з іншими алгоритмами на основі Square-подібної SPN-структури, блоковий шифр «Калина» має наступні істотні конструктивні відмінності:

– початкове й кінцеве забілювання з використанням модульного додавання (264) для підвищення складності криптоаналітичних атак;

– застосування чотирьох різні S-блоків (замість одного) із властивостями для захисту від алгебраїчних атак, і при порівнянні характеристик з іншими блоковими й потоковими шифрами забезпечують найбільшу нелінійність булевих функцій (104), що дає додатковий запас стійкості перетворення;

– збільшений розмір МДР-перетворення, що поліпшує криптографічні властивості й дозволяє оптимізувати швидкодія на сучасних 64-бітових платформах;

– нову односпрямовану схему формування циклових ключів, що забезпечує захист від атак, ефективність програмної й програмно-апаратної реалізації, разом з додатковою стійкістю до методів аналізу спеціального виду.

Оцінка криптографічної стійкості до диференціального, лінійному, алгебраїчному, інтегральному й іншим методам аналізу (практичний критерій) показала, що шифр є стійким при 6 циклах для 128-бітового блоку, 7 циклах для 256-бітового й 9 циклах для 512-бітового (кожний додатковий цикл забезпечує експонентний ріст складності криптоаналізу). Таким чином, шифр, що містить 10, 14 і 18 циклів для блоку 128, 256 і 512 біт відповідно, забезпечує захист від розглянутих видів аналізу й має істотний запас стійкості.

Крім блокового шифру, ДСТУ 7624:2014 визначає режими роботи, що відповідають як ISO 10116:2006, так і додаткові, призначені для сучасних систем

криптографічного захисту IP-трафіка, прозорого шифрування носіїв інформації й ін. У стандарті визначені обсяги повідомлень, після обробки яких потрібна обов'язкова зміна ключа. Крім того, приводяться рекомендації розроблювачам, що обертають увагу на необхідність запобігання атак з використанням особливостей реалізації засобів шифрування. Зокрема, враховані особливості, що дозволяли організацію атак BEAST і CRIME/BREACH у протоколах SSL/TLS, повторне приймання повідомлення, відновлення конфіденційних параметрів на основі залежності часу шифрування від оброблюваних даних (через промахи кешу при табличній реалізації) і ін.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на розділи. У розділах реалізується наступний функціонал:

– З’єднання з мережею.

– Списки та фільтри IP-адрес.

– Кодування за допомогою алгоритму Ріда-Соломона.

– Декодування за допомогою алгоритму Ріда-Соломона.

– Статистика.

– Параметри кодування та шифрування.

Із використанням пунктів Кодування/Декодування визначається початок та завершення процедур кодування, а також виведення детальної інформації щодо кількості правильних та пошкоджених при передачі томів.

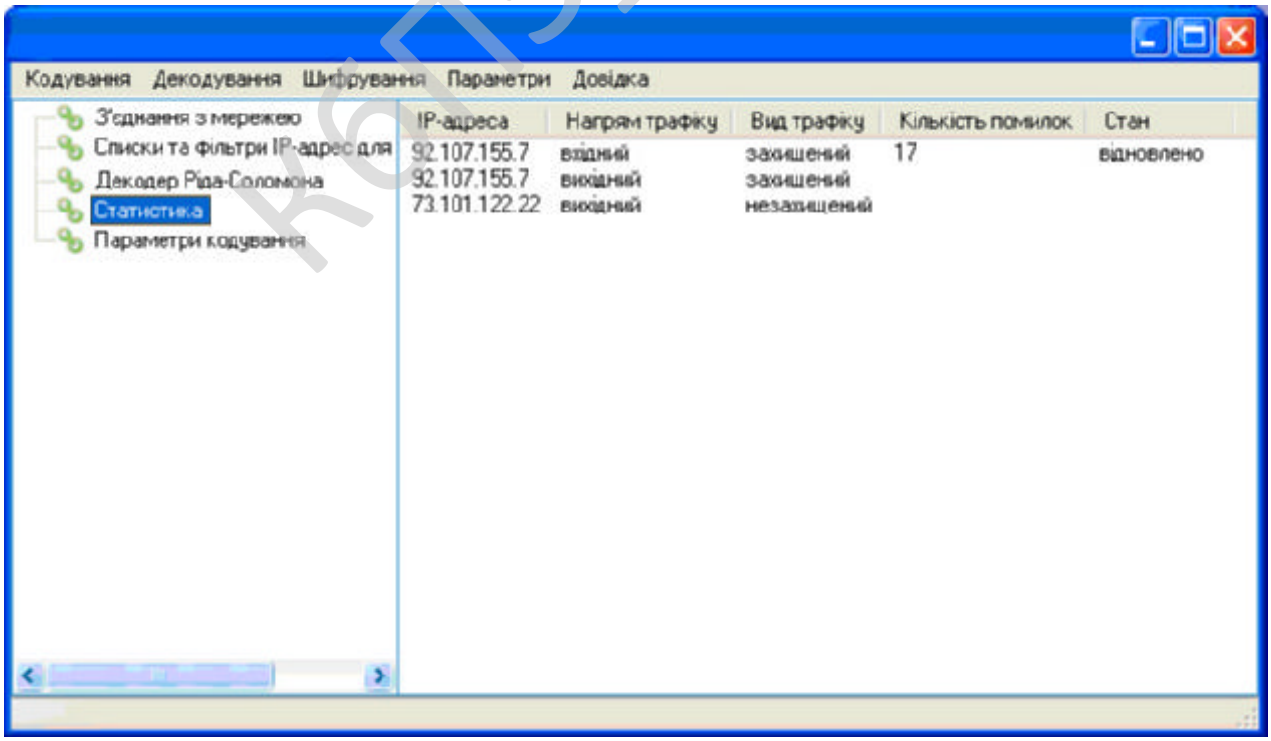

Рисунок 5.1 – Головне вікно ПЗ

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

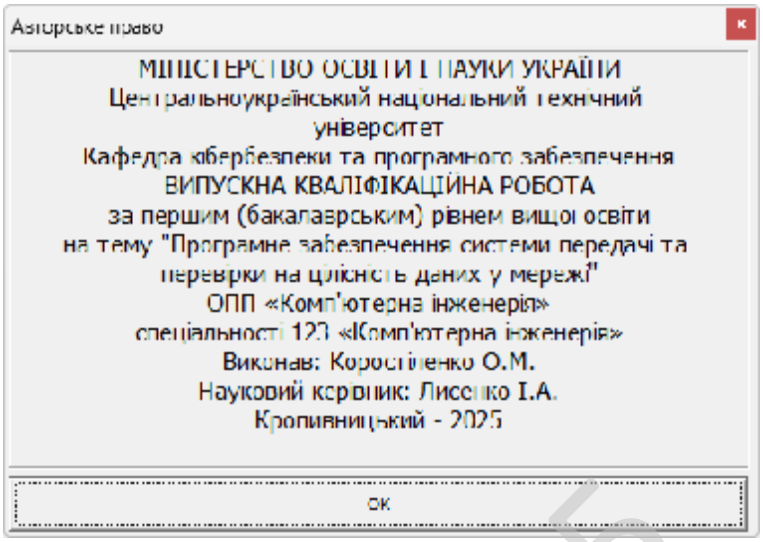

Рисунок 5.2 – Авторське право

Обрано умови розповсюдження – Freeware. Це власницьке програмне забезпечення, котре можна Безоплатно використовувати протягом необмеженого терміну без обмежень у функціональності, і поширюване без сирцевих кодів. Автори такого програмного забезпечення, як правило, хочуть «дати щось спільноті», але хочуть також контролювати його подальшу розробку. Дуже часто плутають поняття «безплатне програмне забезпечення» та «вільне програмне забезпечення», хоча вони суттєво відрізняються.

Безплатне програмне забезпечення можна безоплатно встановлювати та використовувати (іноді з певними обмеженнями, як, наприклад, «безплатне для домашнього або некомерційного вжитку»), в той час як вільне програмне забезпечення можна продавати за будь-яку суму, але при тому, у користувача, котрий його отримує, повинні бути права на вивчення, модифікацію та поширення сирцевих кодів одержаної програми.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи передачі та перевірки на цілісність даних у мережі.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем передачі та перевірки на цілісність даних у мережі.

– Досліджена система передачі та перевірки на цілісність даних у мережі.

– На основі отриманих результатів досліджень створена програмна реалізація системи передачі та перевірки на цілісність даних у мережі.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання передачі та перевірки на цілісність даних у мережі.

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 призначені для системи передачі та перевірки на цілісність даних у мережі. 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 ДСТУ 7624:2014.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Loren Kohnfelder. Designing Secure Software. No Starch Press. 2022. 332 p.

2. Samir Kumar Rakshit. Ethical Hacker's Penetration Testing Guide. BPB Online. 2022. 509 p.

3. Corey J. Ball. Hacking APIs. No Starch Press. 2022. 353 p.

4. Kevin Beaver. Hacking for Dummies. John Wiley & Sons. 2022. 419 p.

5. Mark S. Merkow. Practical Security for Agile and DevOps. CRC Press. 2022. 236 p

6. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

7. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

8. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

9. Lakhno, V., Malyukov, V., Smirnov, O., Bebeshko, B., Chubaievskiy, V., Zhumadilova, M., Malyukova, I., Smirnov, S. «Multifactorial Model for Targeted Attacks Counteracting Within the Framework of a Multi-Step Quality Game with Fuzzy Information». *8th International Symposium on Intelligent Informatics, ISI 2023*, 2025. vol 389. pp 377-389. Springer, Singapore.

10. Kuznetsov, O., Smirnov, O., Mormul, M., Kotukh, Y., Zvieriev, V. «Comparative Research on Cryptocurrency Efficiency: An Objective Analysis of Key Metrics». *International Journal of Computing* 23(4), 2024. pp. 563-573.

11. Kuznetsov O., Frontoni E., Kuznetsova K., Smirnov O., Kostenko V. «Blockchain applications in metaverse environments: new horizons». *Advanced Metaverse Wireless Communication Systems.* pp. 255-293. 2024.

12. Kuznetsov, O., Frontoni, E., Chevardin, V., Smirnov, O., Imoize, A.L. «Advancing metaverse security with cryptographic innovations». *Advanced Metaverse Wireless Communication Systems.* pp 351-386. 2024.

13. Kuznetsov O., Frontoni E., Kuznetsova Y., Smirnov O., Moskovchenko I. «Trust-Based Security Architecture for Edge Computing: A Simulation Study of Dynamic Trust Evolution and Attack Detection». *CEUR Workshop Proceedings,* 2024, 3909, pp. 227–241.

14. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

15. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447.

16. Ткаченко, О., Ільєнко, А., Улічев, О., Мелешко, Є., Смірнов, О. «Правові засади поширення інформаційних впливів в соціальних мережах». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»,* 2024. № 2(26), С. 170–188.

17. 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.

18. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н., Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

19. Kuznetsov, O., Frontoni, E., Kandiy, S., Smirnova, T., Prokopov, S., Bilanovych, A. «New Cost Function for S-boxes Generation by Simulated Annealing

Algorithm». Lecture Notes on Data Engineering and Communications Technologies, 2023. vol 180. pp. 310-320. Springer, Cham.

20. Kuznetsov, O., Frontoni, E., Kandiy, S., Smirnov, O., Ulianovska, Y., Kobylianska, O. «Heuristic Search for Nonlinear Substitutions for Cryptographic Applications». *Lecture Notes on Data Engineering and Communications Technologies,* 2023. vol 180. Springer, Cham. pp. 288-298.

21. Kuznetsov, O., Kuznetsova, Y., Smirnov, O., Kostenko, O., Zvieriev, V. «Evaluating Hashing Algorithms in the Age of ASIC Resistance». *CEUR Workshop Proceedings*, 2023, 3628, pp. 93-105.

22. Kuznetsov O., Frontoni E., Kuznetsova Ye., Smirnov O., Chevardin V. «Achieving Enhanced Security in Biometric Authentication: A Rigorous Analysis of Code-Based Fuzzy Extractor». *CEUR Workshop Proceedings,* Volume 3624, 2023, pp. 330-339.

23. 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.

24. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

25. Смірнов О.А. Козлов Я.О., Смірнова Т.В. «Дослідження застосування SIEM-систем для забезпечення кібербезпеки та захисту інформації». *II Міжнародна науково-практична Інтернет-конференція «Інновації та перспективні шляхи hозвитку інформаційних технологій (ІПШРІТ-2023)»* м.Черкаси 6 грудня 2023 року – Черкаси: ЧДТУ.– 2023. – С.251-252.

26. Козлов Я.О., Смірнова Т.В., Смірнов О.А. «Дослідження SIEM-систем для забезпечення кібербезпеки». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення,* м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 26.

27. Козлов Я.О., Козірова Н.Л., Смірнов О.А. «Дослідження структури та принципу роботи SIEM-системи». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення,* м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 59.

28. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 2(72), С. 170-178.

29. Смірнова Т.В., Гнатюк С.О., Бердибаєв Р.Ш., Сидоренко В.М., Жигаревич О.К., «Система корелювання подій та управління інцидентами кібербезпеки на об'єктах критичної інфраструктури». *Кібербезпека: освіта, наука, техніка*, №*3*(19), 2023, С. 176-196.

30. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022,

31. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies*, vol 131. 2023. Springer, Singapore. pp. 21-34.

32. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

33. 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.

34. 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.

35. 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.

36. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Книшук А.В. «Вступ до кібербезпеки»: навчальний посібник – Кропивницький: ЦНТУ – 2022. – 968 с.

37. Теорія та практика сучасного інформаційно-психологічного протиборства: навчальний посібник / [В.М. Петрик, С.О. Гнатюк, М.М. Присяжнюк та ін.]; за заг. ред. С.О. Гнатюка, В.М. Петрика та О.А Смірнова. – Полтава, 2022. – 334 с.

38. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021,* Cracow, Poland, 22-25 September 2021. P. 414-418

39. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021*, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

40. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

41. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

42. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». *International Journal of Computing*; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

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

44. 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.

45. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

46. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

47. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

48. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum

image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

49. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

50. 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.

51. 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.

52. 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.

53. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». *International Journal of Computer Network and Information Security (IJCNIS)*. Vol. 12, No. 3, 2020. PP.33-43.

54. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 646-660.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи передачі та перевірки на цілісність даних у мережі.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 46-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи передачі та перевірки на цілісність даних у мережі.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи передачі та перевірки на цілісність даних у мережі;

– 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 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 56 аркушів.

8 Етапи розробки

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

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

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 6.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

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

_____ Лисенко І.А.

Програмне забезпечення системи передачі та перевірки на цілісність даних у мережі

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 61

Літера: РП

Кропивницький – 2025 року

Файл BenchmarkForm.cs - вікно тестування швидкодії алгоритму

```
namespace Peredacha_ta_Perevirka_u_Merezhi
{
    partial class BenchmarkForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.coderConfigGroupBox = new System.Windows.Forms.GroupBox();
            this.eccCountLabel = new System.Windows.Forms.Label();
            this.dataCountLabel = new System.Windows.Forms.Label();
            this.eccCountLabel_ = new System.Windows.Forms.Label();
            this.dataCountLabel_ = new System.Windows.Forms.Label();
            this.coderSpeedGroupBox = new System.Windows.Forms.GroupBox();
            this.processedDataCountLabel = new System.Windows.Forms.Label();
            this.processedDataCountLabel_ = new System.Windows.Forms.Label();
            this.timeInTestLabel = new System.Windows.Forms.Label();
            this.timeInTestLabel_ = new System.Windows.Forms.Label();
            this.benchmarkTimer = new
System.Windows.Forms.Timer(this.components);
            this.toolTip = new System.Windows.Forms.ToolTip(this.components);
            this.pauseButtonXP = new PinkieControls.ButtonXP();
            this.closeButtonXP = new PinkieControls.ButtonXP();
            this.closingTimer = new System.Windows.Forms.Timer(this.components);
            this.coderConfigGroupBox.SuspendLayout();
            this.coderSpeedGroupBox.SuspendLayout();
            this.SuspendLayout();
            //
            // coderConfigGroupBox
            //
            this.coderConfigGroupBox.Controls.Add(this.eccCountLabel);
            this.coderConfigGroupBox.Controls.Add(this.dataCountLabel);
            this.coderConfigGroupBox.Controls.Add(this.eccCountLabel_);
            this.coderConfigGroupBox.Controls.Add(this.dataCountLabel_);
            this.coderConfigGroupBox.Location = new System.Drawing.Point(12,
90);
            this.coderConfigGroupBox.Name = "coderConfigGroupBox";
            this.coderConfigGroupBox.Size = new System.Drawing.Size(212, 72);
            this.coderConfigGroupBox.TabIndex = 0;
```

```
            this.coderConfigGroupBox.TabStop = false;
            this.coderConfigGroupBox.Text = "Конфігурація кодера";
            //
            // eccCountLabel
            //
            this.eccCountLabel.AutoSize = true;
            this.eccCountLabel.Location = new System.Drawing.Point(161, 47);
            this.eccCountLabel.Name = "eccCountLabel";
            this.eccCountLabel.Size = new System.Drawing.Size(37, 13);
            this.eccCountLabel.TabIndex = 0;
            this.eccCountLabel.Text = "65535";
            //
            // dataCountLabel
            //
            this.dataCountLabel.AutoSize = true;
            this.dataCountLabel.Location = new System.Drawing.Point(161, 25);
            this.dataCountLabel.Name = "dataCountLabel";
            this.dataCountLabel.Size = new System.Drawing.Size(37, 13);
            this.dataCountLabel.TabIndex = 0;
            this.dataCountLabel.Text = "65535";
            //
            // eccCountLabel_
            //
            this.eccCountLabel_.AutoSize = true;
            this.eccCountLabel_.Location = new System.Drawing.Point(11, 47);
            this.eccCountLabel_.Name = "eccCountLabel_";
            this.eccCountLabel_.Size = new System.Drawing.Size(125, 13);
            this.eccCountLabel_.TabIndex = 0;
            this.eccCountLabel_.Text = "Томів для відновлення:";
            //
            // dataCountLabel_
            //
            this.dataCountLabel_.AutoSize = true;
            this.dataCountLabel_.Location = new System.Drawing.Point(11, 25);
            this.dataCountLabel_.Name = "dataCountLabel_";
            this.dataCountLabel_.Size = new System.Drawing.Size(89, 13);
            this.dataCountLabel_.TabIndex = 0;
            this.dataCountLabel_.Text = "Основних томів:";
            //
            // coderSpeedGroupBox
            //
            this.coderSpeedGroupBox.Controls.Add(this.processedDataCountLabel);
            this.coderSpeedGroupBox.Controls.Add(this.processedDataCountLabel_);
            this.coderSpeedGroupBox.Controls.Add(this.timeInTestLabel);
            this.coderSpeedGroupBox.Controls.Add(this.timeInTestLabel_);
            this.coderSpeedGroupBox.Location = new System.Drawing.Point(12, 9);
            this.coderSpeedGroupBox.Name = "coderSpeedGroupBox";
            this.coderSpeedGroupBox.Size = new System.Drawing.Size(212, 72);
            this.coderSpeedGroupBox.TabIndex = 0;
            this.coderSpeedGroupBox.TabStop = false;
            this.coderSpeedGroupBox.Text = "Швидкість: - Мбайт/з";
            this.coderSpeedGroupBox.Enter += new
System.EventHandler(this.coderSpeedGroupBox_Enter);
            //
            // processedDataCountLabel
            //
            this.processedDataCountLabel.AutoSize = true;
            this.processedDataCountLabel.Location = new System.Drawing.Point(55,
47);
            this.processedDataCountLabel.Name = "processedDataCountLabel";
            this.processedDataCountLabel.Size = new System.Drawing.Size(10, 13);
            this.processedDataCountLabel.TabIndex = 0;
            this.processedDataCountLabel.Text = "-";
            //
            // processedDataCountLabel_
            //
            this.processedDataCountLabel_.AutoSize = true;
            this.processedDataCountLabel_.Location = new
System.Drawing.Point(11, 47);
```

```
            this.processedDataCountLabel_.Name = "processedDataCountLabel_";
            this.processedDataCountLabel_.Size = new System.Drawing.Size(40,
13);
            this.processedDataCountLabel_.TabIndex = 0;
            this.processedDataCountLabel_.Text = "Про\'єм:";
            //
            // timeInTestLabel
            //
            this.timeInTestLabel.AutoSize = true;
            this.timeInTestLabel.Location = new System.Drawing.Point(55, 25);
            this.timeInTestLabel.Name = "timeInTestLabel";
            this.timeInTestLabel.Size = new System.Drawing.Size(10, 13);
            this.timeInTestLabel.TabIndex = 0;
            this.timeInTestLabel.Text = "-";
            //
            // timeInTestLabel_
            //
            this.timeInTestLabel_.AutoSize = true;
            this.timeInTestLabel_.Location = new System.Drawing.Point(11, 25);
            this.timeInTestLabel_.Name = "timeInTestLabel_";
            this.timeInTestLabel_.Size = new System.Drawing.Size(30, 13);
            this.timeInTestLabel_.TabIndex = 0;
            this.timeInTestLabel_.Text = "Година:";
            //
            // benchmarkTimer
            //
            this.benchmarkTimer.Interval = 1000;
            this.benchmarkTimer.Tick += new
System.EventHandler(this.BenchmarkTimer_Tick);
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // pauseButtonXP
            //
            this.pauseButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.pauseButtonXP.DefaultScheme = true;
            this.pauseButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.pauseButtonXP.Hint = "";
            this.pauseButtonXP.Location = new System.Drawing.Point(12, 175);
            this.pauseButtonXP.Name = "pauseButtonXP";
            this.pauseButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.pauseButtonXP.Size = new System.Drawing.Size(101, 23);
            this.pauseButtonXP.TabIndex = 0;
            this.pauseButtonXP.Text = "Пауза";
            this.toolTip.SetToolTip(this.pauseButtonXP, "Постановка/зняття
процесу тестування продуктивності з паузи");
            this.pauseButtonXP.Click += new
System.EventHandler(this.pauseButtonXP_Click);
            //
            // closeButtonXP
            //
            this.closeButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.closeButtonXP.DefaultScheme = true;
            this.closeButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.closeButtonXP.Hint = "";
            this.closeButtonXP.Location = new System.Drawing.Point(123, 175);
            this.closeButtonXP.Name = "closeButtonXP";
            this.closeButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
```

```
            this.closeButtonXP.Size = new System.Drawing.Size(101, 23);
            this.closeButtonXP.TabIndex = 1;
            this.closeButtonXP.Text = "Закрити";
            this.toolTip.SetToolTip(this.closeButtonXP, "Припинення тестування
продуктивності із закриттям даного вікна");
            this.closeButtonXP.Click += new
System.EventHandler(this.closeButtonXP_Click);
            //
            // closingTimer
            //
            this.closingTimer.Tick += new
System.EventHandler(this.closingTimer_Tick);
            //
            // BenchmarkForm
            //
            this.AutoScaleDimensions = new System.Drawing.Size(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(236, 210);
            this.ControlBox = false;
            this.Controls.Add(this.pauseButtonXP);
            this.Controls.Add(this.closeButtonXP);
            this.Controls.Add(this.coderSpeedGroupBox);
            this.Controls.Add(this.coderConfigGroupBox);
            this.DoubleBuffered = true;
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "BenchmarkForm";
            this.ShowIcon = false;
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "Підготовка";
            this.Load += new System.EventHandler(this.BenchmarkForm_Load);
            this.coderConfigGroupBox.ResumeLayout(false);
            this.coderConfigGroupBox.PerformLayout();
            this.coderSpeedGroupBox.ResumeLayout(false);
            this.coderSpeedGroupBox.PerformLayout();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox coderConfigGroupBox;
        private System.Windows.Forms.Label dataCountLabel_;
        private System.Windows.Forms.Label eccCountLabel_;
        private System.Windows.Forms.Label eccCountLabel;
        private System.Windows.Forms.Label dataCountLabel;
        private System.Windows.Forms.GroupBox coderSpeedGroupBox;
        private System.Windows.Forms.Label timeInTestLabel_;
        private System.Windows.Forms.Label timeInTestLabel;
        private System.Windows.Forms.Label processedDataCountLabel;
        private System.Windows.Forms.Label processedDataCountLabel_;
        private System.Windows.Forms.Timer benchmarkTimer;
        private PinkieControls.ButtonXP closeButtonXP;
        private PinkieControls.ButtonXP pauseButtonXP;
        private System.Windows.Forms.ToolTip toolTip;
        private System.Windows.Forms.Timer closingTimer;
    }
}
```

Файл About.cs - вікно довідки про програму

```
namespace Peredacha_ta_Perevirka_u_Merezhi
{
    partial class About
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(About));
            this.button1 = new System.Windows.Forms.Button();
            this.label1 = new System.Windows.Forms.Label();
            this.pictureBox1 = new System.Windows.Forms.PictureBox();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.label6 = new System.Windows.Forms.Label();
            this.label7 = new System.Windows.Forms.Label();
            this.label8 = new System.Windows.Forms.Label();

((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(346, 229);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(92, 23);
            this.button1.TabIndex = 1;
            this.button1.Text = "OK";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif",
8.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.label1.Location = new System.Drawing.Point(11, 12);
            this.label1.Name = "label1";
```

```
            this.label1.Size = new System.Drawing.Size(144, 13);
            this.label1.TabIndex = 2;
            this.label1.Text = "ДИПЛОМНИЙ ПРОЕКТ";
            //
            // pictureBox1
            //
            this.pictureBox1.Image =
((System.Drawing.Image)(resources.GetObject("pictureBox1.Image")));
            this.pictureBox1.Location = new System.Drawing.Point(305, -28);
            this.pictureBox1.Name = "pictureBox1";
            this.pictureBox1.Size = new System.Drawing.Size(133, 132);
            this.pictureBox1.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.Zoom;
            this.pictureBox1.TabIndex = 0;
            this.pictureBox1.TabStop = false;
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Location = new System.Drawing.Point(12, 45);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(49, 13);
            this.label2.TabIndex = 3;
            this.label2.Text = "на тему:";
            //
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.Location = new System.Drawing.Point(12, 79);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(257, 13);
            this.label3.TabIndex = 4;
            this.label3.Text = "Програмне забезпечення  ";
            //
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.Location = new System.Drawing.Point(12, 102);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(349, 13);
            this.label4.TabIndex = 5;
            this.label4.Text = " системи передачі та перевірки на цілісність
даних у мережі ";
            //
            // label5
            //
            this.label5.AutoSize = true;
            this.label5.Location = new System.Drawing.Point(16, 140);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(139, 13);
            this.label5.TabIndex = 6;
            this.label5.Text = "Автор: студент Татчук В.Я.";
            //
            // label6
            //
            this.label6.AutoSize = true;
            this.label6.Location = new System.Drawing.Point(54, 163);
            this.label6.Name = "label6";
            this.label6.Size = new System.Drawing.Size(63, 13);
            this.label6.TabIndex = 7;
            this.label6.Text = "гр. КІ(ПМ)-14С";
            //
            // label7
            //
            this.label7.AutoSize = true;
            this.label7.Location = new System.Drawing.Point(16, 194);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(128, 13);
            this.label7.TabIndex = 8;
```

```
            this.label7.Text = "Керівник: Минайленко Р.М.";
            //
            // label8
            //
            this.label8.AutoSize = true;
            this.label8.Location = new System.Drawing.Point(16, 233);
            this.label8.Name = "label8";
            this.label8.Size = new System.Drawing.Size(104, 13);
            this.label8.TabIndex = 9;
            this.label8.Text = "м. Кіровоград 2015";
            //
            // About
            //
            this.AutoScaleDimensions = new System.Drawing.Size(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(445, 258);
            this.Controls.Add(this.label8);
            this.Controls.Add(this.label7);
            this.Controls.Add(this.label6);
            this.Controls.Add(this.label5);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.button1);
            this.Controls.Add(this.pictureBox1);
            this.Name = "About";
            this.Text = "Про програму...";

((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.PictureBox pictureBox1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.Label label7;
        private System.Windows.Forms.Label label8;
    }
}
```

Кодер Ріда-Соломона для системи передачі та перевірки на цілісність даних у мережі

Файл RSRaidEncoder.cs - кодування алгоритмом Ріда-Соломона для системи передачі та перевірки на цілісність даних у мережі

```
using System;
using System.Threading;

namespace Peredacha_ta_Perevirka_u_Merezhi
{
    /// <summary>
    /// Клас RAID-подібного кодера Ріда-Соломона для системи передачі та
перевірки на цілісність даних у мережі
    /// </summary>
    public class RSRaidEncoder : RSRaidBase
    {
        #region Construction & Destruction

        /// <summary>
        /// Конструктор кодера за замовчуванням
        /// </summary>
        public RSRaidEncoder()
        {
            // 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 RSRaidEncoder(int dataCount, int eccCount)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, (int)RSType.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 кодека Ріда-Соломона для системи
передачі та перевірки на цілісність даних у мережі (по типу матриці)</param>
        public RSRaidEncoder(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 Ріда-Соломона для системи
передачі та перевірки на цілісність даних у мережі (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)RSType.Dispersal)
            {
                maxVolCount = (int)RSConst.MaxVolCountDisp;

            } else
            {
                maxVolCount = (int)RSConst.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.eRSType)
                    )
                {
                    this.mainConfigChanged = true;
                }

                // Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eRSType = 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.eRSType == (int)RSType.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.eRSType == (int)RSType.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 декодера беремо дані з відповідного
масиву
                    if (this.eRSType == (int)RSType.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 (OnRSMatrixFormingFinish != null)
                {
                    //...повідомляємо, що екземпляр класу готовий до роботи
                    OnRSMatrixFormingFinish();
                }

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

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

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

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

        #endregion Private Operations
    }
}
```

Декодер Ріда-Соломона для системи передачі та перевірки на цілісність даних у мережі

Файл RSRaidDecoder.cs - декодування алгоритмом Ріда-Соломона для системи передачі та перевірки на цілісність даних у мережі

```
using System;
using System.Threading;

namespace Peredacha_ta_Perevirka_u_Merezhi
{
    /// <summary>
    /// Клас RAID-подібного декодера Ріда-Соломона для системи передачі та
перевірки на цілісність даних у мережі
    /// </summary>
    public class RSRaidDecoder : RSRaidBase
    {
        #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 RSRaidDecoder()
        {
            // 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 RSRaidDecoder(int dataCount, int eccCount, int[] volList)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList, (int)RSType.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 кодека Ріда-Соломона для системи
передачі та перевірки на цілісність даних у мережі (по типу матриці)</param>
        public RSRaidDecoder(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 Ріда-Соломона для системи
передачі та перевірки на цілісність даних у мережі (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)RSType.Dispersal)
            {
                maxVolCount = (int)RSConst.MaxVolCountDisp;

            } else
            {
                maxVolCount = (int)RSConst.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.eRSType)
                    )
                {
                    this.mainConfigChanged = true;
                }

                // Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eRSType = 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.eRSType == (int)RSType.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
зворотної
                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 і-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)
                     &&
                        (OnUpdateRSMatrixFormingProgress != null)
                   )
                {
                    //...виводимо дані
                    OnUpdateRSMatrixFormingProgress((((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.eRSType == (int)RSType.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.eRSType == (int)RSType.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 (OnRSMatrixFormingFinish != null)
            {
                //...повідомляємо, що екземпляр класу готовий до роботи
                OnRSMatrixFormingFinish();
            }

            // 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 FileAnalyzer.cs - контроль цілісності набору файлів-томів

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

namespace Peredacha_ta_Perevirka_u_Merezhi
{
    /// <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 кодека Ріда-Соломона для системи передачі та перевірки на
цілісність даних у мережі (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 кодека Ріда-Соломона для системи
передачі та перевірки на цілісність даних у мережі (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)RSConst.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 Ріда-Соломона для системи передачі та
перевірки на цілісність даних у мережі (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 кодека Ріда-Соломона для системи
передачі та перевірки на цілісність даних у мережі (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)RSConst.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 Ріда-Соломона для системи передачі та
перевірки на цілісність даних у мережі (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)
                        {
                            //...exitимо 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)
                                {
                                    //...exitимо 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)
                            {
                                //...exitимо 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 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
        }
    }
```

Інтерфейс користувача

Файл MainForm.cs - головні вікно програми

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

        /// <summary>
        ///
        /// </summary>
        /// <param name="disposing">правда, якщо керуючі ресурси повині бути
розташовані, неправда у 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("Downloads", 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);
            //
            // quickextractionToolStripMenuItem
            //
            this.quickextractionToolStripMenuItem.Name =
"quickextractionToolStripMenuItem";
            this.quickextractionToolStripMenuItem.Size = new
System.Drawing.Size(216, 22);
            this.quickextractionToolStripMenuItem.Text = "Швидке вилучення
диску";
            this.quickextractionToolStripMenuItem.Click += new
System.EventHandler(this.quickextractionToolStripMenuItem_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.ForeColor =
System.Drawing.SystemColors.ControlText;
            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 кодера (основних
томів: ...; томів для відновлення: ...; обсяг виход" +
                "в: ...)";
            //
            // redundancyGroupBox
            //
            this.redundancyGroupBox.Controls.Add(this.redundancyMacTrackBar);
            this.redundancyGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.redundancyGroupBox.ForeColor =
System.Drawing.SystemColors.ControlText;
            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 = "Надмірність кодування:";
            //
            // 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.toolTip.SetToolTip(this.redundancyMacTrackBar, "Чим більше
надмірність кодування - том вище відмовостійкість, алі більше й обсяг " +
                    "отриманого набору томів");
            this.redundancyMacTrackBar.TrackerColor =
System.Drawing.Color.RoyalBlue;
            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);
            //
            // allVolCountGroupBox
            //
            this.allVolCountGroupBox.Controls.Add(this.allVolCountMacTrackBar);
            this.allVolCountGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.allVolCountGroupBox.ForeColor =
System.Drawing.SystemColors.ControlText;
            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 = "Загальна кількість томів:";
            //
            // 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 = 24;
            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.toolTip.SetToolTip(this.allVolCountMacTrackBar, "Чим більше
томів - том повільніше обробка й вище відмовостійкість");
            this.allVolCountMacTrackBar.TrackerColor =
System.Drawing.Color.RoyalBlue;
            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 = 14;
            this.allVolCountMacTrackBar.ValueChanged += new
EConTech.Windows.MACUI.ValueChangedHandler(this.allVolCountMacTrackBar_ValueChan
ged);
            this.allVolCountMacTrackBar.MouseUp += new
System.Windows.Forms.MouseEventHandler(this.allVolCountMacTrackBar_MouseUp);
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // browser
            //
            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 = "Documents and Settings";
            treeNode2.SelectedImageIndex = 20;
            treeNode2.Text = "Documents and Settings";
            treeNode3.Name = "";
            treeNode3.Text = "";
            treeNode4.ImageIndex = 18;
            treeNode4.Name = "Downloads";
            treeNode4.SelectedImageIndex = 20;
            treeNode4.Text = "Downloads";
            treeNode5.Name = "";
            treeNode5.Text = "";
            treeNode6.ImageIndex = 18;
            treeNode6.Name = "Program Files";
            treeNode6.SelectedImageIndex = 20;
            treeNode6.Text = "Program Files";
            treeNode7.Name = "";
            treeNode7.Text = "";
            treeNode8.ImageIndex = 18;
            treeNode8.Name = "RapidDriver";
            treeNode8.SelectedImageIndex = 20;
            treeNode8.Text = "RapidDriver";
            treeNode9.Name = "";
            treeNode9.Text = "";
            treeNode10.ImageIndex = 18;
            treeNode10.Name = "Server";
            treeNode10.SelectedImageIndex = 20;
            treeNode10.Text = "Server";
            treeNode11.Name = "";
            treeNode11.Text = "";
```

```
            treeNode12.ImageIndex = 18;
            treeNode12.Name = "WINDOWS";
            treeNode12.SelectedImageIndex = 20;
            treeNode12.Text = "WINDOWS";
            treeNode13.Name = "";
            treeNode13.Text = "";
            treeNode14.ImageIndex = 18;
            treeNode14.Name = "WINDOWS.0";
            treeNode14.SelectedImageIndex = 20;
            treeNode14.Text = "WINDOWS.0";
            treeNode15.ImageIndex = 23;
            treeNode15.Name = "SYSTEM2 (C:)";
            treeNode15.SelectedImageIndex = 24;
            treeNode15.Text = "SYSTEM2 (C:)";
            this.browser.SelectedNode = treeNode15;
            this.browser.ShowFoldersButton = false;
            this.browser.ShowNavigationBar = false;
            this.browser.Size = new System.Drawing.Size(962, 432);
            this.browser.SplitterDistance = 398;
            this.browser.StartUpDirectory = FileBrowser.SpecialFolders.Other;
            this.browser.StartUpDirectoryOther = "C:\\";
            this.browser.TabIndex = 0;
            //
            // repairButton
            //
            this.repairButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.repairButton.Image =
((System.Drawing.Image)(resources.GetObject("repairButton.Image")));
            this.repairButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.repairButton.Location = new System.Drawing.Point(308, 27);
            this.repairButton.Name = "repairButton";
            this.repairButton.Size = new System.Drawing.Size(100, 97);
            this.repairButton.TabIndex = 3;
            this.repairButton.Text = "Виправити";
            this.repairButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.toolTip.SetToolTip(this.repairButton, "Відновити цілісність
відмовостійкого набору томів");
            this.repairButton.UseVisualStyleBackColor = true;
            this.repairButton.Click += new
System.EventHandler(this.repairButton_Click);
            //
            // testButton
            //
            this.testButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.testButton.Image =
((System.Drawing.Image)(resources.GetObject("testButton.Image")));
            this.testButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.testButton.Location = new System.Drawing.Point(210, 27);
            this.testButton.Name = "testButton";
            this.testButton.Size = new System.Drawing.Size(100, 97);
            this.testButton.TabIndex = 4;
            this.testButton.Text = "Перевірити";
            this.testButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.toolTip.SetToolTip(this.testButton, "Перевірити на наявність
помилок відмовостійкий набір томів");
            this.testButton.UseVisualStyleBackColor = true;
            this.testButton.Click += new
System.EventHandler(this.testButton_Click);
            //
            // recoverButton
            //
            this.recoverButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.recoverButton.Image =
((System.Drawing.Image)(resources.GetObject("recoverButton.Image")));
```

```
            this.recoverButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.recoverButton.Location = new System.Drawing.Point(111, 27);
            this.recoverButton.Name = "recoverButton";
            this.recoverButton.Size = new System.Drawing.Size(100, 97);
            this.recoverButton.TabIndex = 2;
            this.recoverButton.Text = "Декодувати";
            this.recoverButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.toolTip.SetToolTip(this.recoverButton, "Розкодувати файли з
набору томів з корекцією помилок");
            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 =
((System.Drawing.Image)(resources.GetObject("protectButton.Image")));
            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 = "Кодувати";
            this.protectButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.toolTip.SetToolTip(this.protectButton, "Закодувати файли у
відмовостійкому форматі в даній директорії");
            this.protectButton.UseVisualStyleBackColor = false;
            this.protectButton.Click += new
System.EventHandler(this.protectButton_Click);
            //
            // exitToolStripMenuItem
            //
            this.exitToolStripMenuItem.Image =
global::Peredacha_ta_Perevirka_u_Merezhi.Properties.Resources.Exit;
            this.exitToolStripMenuItem.Name = "виходToolStripMenuItem";
            this.exitToolStripMenuItem.Size = new System.Drawing.Size(112, 22);
            this.exitToolStripMenuItem.Text = "Вихід";
            this.exitToolStripMenuItem.Click += new
System.EventHandler(this.exitToolStripMenuItem_Click);
            //
            // testspeedToolStripMenuItem
            //
            this.testspeedToolStripMenuItem.Image =
global::Peredacha_ta_Perevirka_u_Merezhi.Properties.Resources.StartBenchmark;
            this.testspeedToolStripMenuItem.ImageScaling =
System.Windows.Forms.ToolStripItemImageScaling.None;
            this.testspeedToolStripMenuItem.Name =
"тестбистродействияToolStripMenuItem";
            this.testspeedToolStripMenuItem.Size = new System.Drawing.Size(161,
22);
            this.testspeedToolStripMenuItem.Text = "Тест швидкодії";
            this.testspeedToolStripMenuItem.Click += new
System.EventHandler(this.testspeedToolStripMenuItem_Click);
            //
            // MainForm
            //
            this.AutoScaleDimensions = new System.Drawing.Size(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.repairButton);
            this.Controls.Add(this.testButton);
            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.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 fileToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem helpToolStripMenuItem;
        private System.Windows.Forms.Button protectButton;
        private System.Windows.Forms.Button recoverButton;
        private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
        private System.Windows.Forms.ToolStripSeparator
separatorToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem aboutToolStripMenuItem;
        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
instrumentToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem
testspeedToolStripMenuItem;
        internal FileBrowser.Browser browser;
        private System.Windows.Forms.ToolStripMenuItem
adjustmentToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem
encodingfilterToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem
quickextractionToolStripMenuItem;
    }
}
```

Файл ProcessForm.cs - вікно кодування/декодування

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

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </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.stopButtonXP = new PinkieControls.ButtonXP();
            this.pauseButtonXP = new PinkieControls.ButtonXP();
            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 томів 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 томів:";
            //
            // 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;
            //
            // stopButtonXP
            //
            this.stopButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.stopButtonXP.DefaultScheme = true;
            this.stopButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.stopButtonXP.Hint = "";
            this.stopButtonXP.Location = new System.Drawing.Point(762, 257);
            this.stopButtonXP.Name = "stopButtonXP";
            this.stopButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.stopButtonXP.Size = new System.Drawing.Size(121, 23);
            this.stopButtonXP.TabIndex = 2;
            this.stopButtonXP.Text = "Перервати обробку";
            this.toolTip.SetToolTip(this.stopButtonXP, "Припинення обробки
файлів із закриттям даного вікна");
            this.stopButtonXP.Click += new
System.EventHandler(this.stopButtonXP_Click);
            //
            // pauseButtonXP
            //
            this.pauseButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.pauseButtonXP.DefaultScheme = true;
            this.pauseButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.pauseButtonXP.Hint = "";
            this.pauseButtonXP.Location = new System.Drawing.Point(762, 220);
            this.pauseButtonXP.Name = "pauseButtonXP";
            this.pauseButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.pauseButtonXP.Size = new System.Drawing.Size(121, 23);
            this.pauseButtonXP.TabIndex = 1;
            this.pauseButtonXP.Text = "Пауза";
            this.toolTip.SetToolTip(this.pauseButtonXP, "Постановка/зняття
процесу обробки з паузи");
            this.pauseButtonXP.Click += new
System.EventHandler(this.pauseButtonXP_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.stopButtonXP);
            this.Controls.Add(this.pauseButtonXP);
            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.ButtonXP pauseButtonXP;
        private PinkieControls.ButtonXP stopButtonXP;
        private System.Windows.Forms.Timer processTimer;
    }
}
```