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 підвищення надійності гібридних систем зберігання інформації SSD/HDD”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Андріяшевський І.В.
«____» _____ 2025 р.

Керівник проекту
доктор технічних наук, професор
_____ Коваленко О.В.
«____» _____ 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

Андріяшевському Івану Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD*

2. Керівник роботи *Коваленко Олександр Володимирович, докт. техн. наук, професор*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 47-02 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 підвищення надійності гібридних систем зберігання інформації SSD/HDD*

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
_____ Коваленко О.В.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Андріяшевський І.В.
(прізвище та ініціали)

АНОТАЦІЯ

Андріяшевський І.В. Програмне забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD. 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 підвищення надійності гібридних систем зберігання інформації SSD/HDD.

Метою розробки є програмне забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD.

Результат роботи – програмна реалізація системи підвищення надійності гібридних систем зберігання інформації SSD/HDD.

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, надійність, SSD/HDD

ABSTRACT

Andriyashevsky I.V. Software for the system for increasing the reliability of hybrid information storage systems SSD/HDD. 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 for increasing the reliability of hybrid information storage systems SSD/HDD.

The purpose of the development is the software for the system for increasing the reliability of hybrid information storage systems SSD/HDD.

The result of the work is the software implementation of the system for increasing the reliability of hybrid information storage systems SSD/HDD.

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, reliability, SSD/HDD

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ОС	–	Операційна система
ПЗ	–	Програмне забезпечення
BIOS	–	Basic Input-Output System
LINQ	–	Language Integrated Query
RAID	–	Redundant Array of Ine10/11ensive Disks

ВСТУП

Актуальність теми. Як відомо, твердотільні накопичувачі SSD, що одержують усе більше широке поширення, набагато перевершують по продуктивності традиційні жорсткі диски HDD. Однак їхня вартість значно вище, і тому використання одних лише SSD як корпоративний пул ресурсів зберігання даних виявляється для більшості компаній невигідним.

«Кращою практикою» для багатьох корпоративних замовників є застосування гібридних систем зберігання SSD/HDD. Таке рішення дозволяє скористатися перевагами обох типів носіїв – великою ємністю HDD і високою швидкодією SSD в IOPS (кількість операцій вводу-виводу в секунду), – але при цьому залишається економічно привабливим.

У гібридній системі зберігання SSD/HDD основна ємність представлена недорогими жорсткими дисками, а невеликий пул для «гарячих», часто використовуваних даних – флеш-пам'ятю. У раціонально спроектованої гібридної СЗД при невеликій кількості накопичувачів SSD досягається значне прискорення операцій з основним пулом зберігання даних.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD.

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

– Огляд існуючих систем підвищення надійності гібридних систем зберігання інформації SSD/HDD.

– Дослідження системи підвищення надійності гібридних систем зберігання інформації SSD/HDD.

– Програмна реалізація системи підвищення надійності гібридних систем зберігання інформації SSD/HDD.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі підвищення надійності гібридних систем зберігання інформації SSD/HDD.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD, 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

У світі цифрових даних фраза «передбачуваний збій SMART на жорсткому диску» є зловісним попередженням про те, що ваш жорсткий диск може доживати останні дні, а безпека ваших даних і стабільність усієї вашої системи знаходяться під загрозою.

Традиційні методи моніторингу стану диска, такі як використання виключно атрибутів SMART, можуть зробити вас вразливими до неочікуваних збоїв диска. Дослідження показали, що понад 30% збоїв диска відбуваються без спрацьовування будь-яких сповіщень про помилки SMART. За допомогою розробленого у даній роботі програмного забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD, ви отримуєте детальну інформацію про стан та продуктивність вашого диска, що дозволяє вам завчасно планувати заміну до того, як станеться лихо.

Відстежуючи необроблені дані про стан диска та системні події, розроблене у даній роботі програмного забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD, щодня завантажує цю інформацію до хмарного механізму штучного інтелекту, надаючи достатньо даних для аналізу збоїв диска. Протягом 24 годин ви отримуватимете сповіщення про потенційні збої диска протягом наступних шести місяців. Це завчасне попередження дозволяє вам вживати превентивних заходів та мінімізує ризик втрати даних і простою системи.

Турбуєтеся про конфіденційність? Розроблене у даній роботі програмного забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD, подбає про вас. Ми не збираємо контент, окрім показників стану дисків та ідентифікаторів користувачів і пристроїв, необхідних для надання

послуг. Доступна опція перенаправлення всіх даних що залишають вашу мережу, через проксі-сервер. Крім того, ви можете налаштувати графік завантаження, щоб гарантувати, що жодні критично важливі дані не будуть пропущені, навіть під час сплячого режиму або простою системи.

1.2 Область застосування

Незалежно від того, чи надаєте ви перевагу хмарному керуванню, чи моніторингу робочого столу, розробленого у даній роботі програмного забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD, пропонує гнучкість, що відповідає вашим потребам. Портал надає зручний інтерфейс для моніторингу стану вашої системи через Інтернет. Для IT-фахівців, які керують кількома пристроями на різних платформах, для Windows® та macOS® пропонує централізовані можливості моніторингу та управління.

Для користувачів, у разі потенційного виходу з ладу диска, розробленого у даній роботі програмного забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD, безперешкодно інтегрується з системами, щоб ініціювати автоматичну заміну дисків RAID, перш ніж дані стануть недоступними. Такий проактивний підхід усуває необхідність тривалих процесів відновлення RAID, зберігає надійність системи та мінімізує перебої в роботі.

Розроблене у даній роботі програмне забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD, не розрізняє типи накопичувачів. Незалежно від того, чи використовуєте ви жорсткі диски/SSD SATA, високопродуктивні жорсткі диски/SSD SAS чи SSD NVMe, розроблене у даній роботі програмне забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD, подбає про вас. Прогнозуючи збої в

широкому спектрі технологій накопичувачів, це комплексне рішення гарантує постійний захист критично важливих бізнес-даних.

Прогнозування та запобігання збоям SMART є життєво важливим у швидкозмінному світі цифрових сховищ. За допомогою розробленого у даній роботі програмного забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD ви можете бути на крок попереду потенційних збоїв дисків, захистити свої дані та забезпечити безперебійну роботу ваших систем.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD, 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

Існує кілька безкоштовних програм, які допоможуть вам перевірити жорсткий диск і виявити будь-які проблеми. Ось мої найкращі рекомендації, які я пропоную людям уже багато років.

Більшість програм мають схожі функції, але деякі працюють лише на певних жорстких дисках. Незалежно від того, яке програмне забезпечення ви використовуєте, вам може знадобитися замінити жорсткий диск, якщо він не пройде якусь частину одного з цих тестів.

Ваш ПК з Windows містить такі інструменти, як перевірка помилок Windows та команда chkdsk . Однак інші, як-от наведені нижче, доступні від виробників жорстких дисків та інших розробників.

Seagate SeaTools

Переваги:

– Працює як зсередини, так і ззовні Windows.

– Повністю безкоштовно.

– SeaTools для DOS працює на будь-якій операційній системі.

– Інструмент Windows дозволяє протестувати жорсткий диск будь-якого виробника.

Недоліки:

– SeaTools для DOS може бути складним у використанні та встановленні.

– DOS-версія може обробляти лише 100 помилок перед перезапуском.

– Версія для DOS може не працювати належним чином з RAID-контролерами.

Огляд Seagate SeaTools

Seagate SeaTools – це безкоштовне програмне забезпечення для тестування жорстких дисків, яке доступне у двох варіантах для домашніх користувачів:

– Завантажувальні програми SeaTools та SeaTools для DOS підтримують накопичувачі Seagate або Maxtor і працюють незалежно від операційної системи на окремому USB- накопичувачі або компакт-диску відповідно.

– SeaTools для Windows встановлюється на вашу систему Windows. Використовуйте його для виконання базового та розширеного тестування будь-якого типу накопичувача – внутрішнього чи зовнішнього – від будь-якого виробника.

Програми Seagate чудові. Вони використовуються для тестування жорстких дисків професійними комп'ютерними сервісами, але достатньо прості для використання будь-ким.

Версія для Windows має працювати з Windows 11, 10, 8, 7, Vista та XP. Також існує версія для Linux.

Ті, хто шукає SeaTools Desktop, SeaTools Online або програмне забезпечення Maxtor PowerMax, зверніть увагу, що вищезгадані інструменти замінили всі три. Тепер бренд Maxtor належить Seagate.

Вбудована перевірка помилок Microsoft Windows

Переваги:

– Завантаження не потрібне.

– Також намагається виправити будь-які помилки жорсткого диска.

Недоліки:

– Працює лише у Windows.

– Бракує функцій, які є в іншому безкоштовному програмному забезпеченні для тестування жорстких дисків.

Перевірка на помилки, яку іноді називають Scandisk , – це інструмент для тестування жорсткого диска, що постачається з Microsoft Windows, який може сканувати ваш жорсткий диск у пошуках широкого спектру помилок та

виправляти багато з них.

Він вбудований у всі сучасні версії Windows.

GSmartControl

Переваги:

– Оберіть один з трьох різних тестів.

– Працює на Windows, Linux та macOS.

– Дозволяє переглянути атрибути SMART диска.

– Доступна портативна версія.

Недоліки:

– Не підтримує всі USB- та RAID-пристрої.

– Під час експорту інформації вона включає все , а не лише конкретний результат, який ви хочете зберегти.

GSmartControl може проводити різні тести жорстких дисків з детальними результатами та надавати загальну оцінку стану диска.

Переглядайте та зберігайте значення атрибутів SMART, такі як кількість циклів увімкнення/вимкнення живлення, коефіцієнт помилок у кількох зонах, кількість спроб калібрування та багато інших.

GSmartControl виконує три самотести для виявлення несправностей диска: короткий самотест триває близько 2 хвилин і використовується для виявлення повністю пошкодженого жорсткого диска, розширений самотест триває 70 хвилин і перевіряє всю поверхню жорсткого диска на наявність несправностей, а самотест під час транспортування – це 5-хвилинний тест, який має на меті виявити пошкодження, що виникли під час транспортування диска.

Цю програму можна завантажити для Windows як портативну програму або як звичайну програму зі звичайним інсталятором. Найновіша версія працює з Windows 11, 10, 8, 7 та Vista, але існує застаріла версія, яку можна отримати для старіших версій Windows. Вона також доступна для операційних систем Linux та Mac, а також включена до кількох програм LiveCD/LiveUSB.

HDDScan

Переваги:

– Працює з усіма жорсткими дисками.

– Не потребує встановлення (портативний).

– Легкий у використанні.

– Включає SMART-тест.

– Працює на всіх сучасних версіях Windows.

Недоліки:

– Працює лише у Windows.

– Немає вбудованої довідкової документації чи порад щодо його використання.

– Не вдається встановити програму на комп'ютер (вона автоматично запускається як портативна програма).

Огляд HDDScan

HDDScan – це безкоштовна програма для тестування жорстких дисків усіх типів, незалежно від виробника. Вона включає SMART -тест та поверхневий тест.

Деякі причини, чому я обрав цей додаток, полягають у тому, що він простий у використанні, повністю портативний, підтримує майже всі інтерфейси накопичувачів і, здається, регулярно оновлюється.

Офіційні системні вимоги вказують, що вам потрібна ОС Windows XP, тому вона має працювати з Windows 11, 10, 8, 7 та Vista, включаючи Windows Server 2003.

Samsung HUTIL

Переваги:

– Тестує жорсткі диски незалежно від встановленої ОС.

– Не надто складний у використанні.

– Також дозволяє стирати дані з диска.

Недоліки:

– Тестує лише жорсткі диски Samsung.

– Не так просто встановити, як програму для настільного комп'ютера.

– Для налаштування програми потрібен доступ до робочого комп'ютера.

– Текстовий інтерфейс (немає кнопок, які можна натискати).

– Не оновлювався багато років.

Огляд Samsung HUTIL

Samsung HUTIL – це безкоштовна утиліта для діагностики жорстких дисків Samsung. HUTIL іноді називають ES-Tool.

Цей інструмент доступний у вигляді ISO-образу для запису на компакт-диск або USB-накопичувач. Ця функція робить HUTIL незалежним від операційної системи та загалом кращим інструментом для тестування, ніж ті, що призначені для використання у Windows. Також можна запустити HUTIL із завантажувальної дискети.

Оскільки це завантажувальна програма, вам знадобиться робочий жорсткий диск та ОС, щоб записати її на диск або USB-пристрій.

HUTIL тестуватиме лише жорсткі диски Samsung. Він завантажить і знайде ваш диск не від Samsung, але ви не зможете виконати жодної діагностики на диску.

Western Digital Dashboard

Переваги:

– Працює зсередини Windows.

– Дійсно простий у використанні.

– Також відображається основна інформація про накопичувачі.

Недоліки:

– Сканує лише жорсткі диски Western Digital.

Огляд панелі інструментів Western Digital

Western Digital Dashboard – це безкоштовне програмне забезпечення для тестування жорстких дисків для Windows, яке дозволяє виконувати кілька тестів жорсткого диска.

Він підтримує перегляд інформації технології самоконтролю, аналізу та

звітності (SMART). Є опція короткого тесту, яка виконує відносно швидке самосканування, та розширеного тесту, який перевіряє весь жорсткий диск на наявність пошкоджених секторів.

Цей інструмент також можна використовувати для очищення жорсткого диска за допомогою методу очищення даних Write Zero .

HD Tune

Переваги:

– Тестує кілька типів пристроїв зберігання даних.

– Включає корисні тести.

– Інформацію можна зберегти на скріншоті та скопіювати в буфер обміну.

– Програма не викликає труднощів у використанні.

Недоліки:

– Не вдається експортувати результати в текстовий файл.

– Офіційно працює лише на Windows до 7.

– Дозволено лише домашнє/особисте використання.

– Останнє оновлення у 2008 році.

HD Tune – це тестер драйверів жорсткого диска на базі Windows, який працює з будь-яким внутрішнім або зовнішнім жорстким диском , SSD або картою пам'яті.

Ви можете запустити тест читання за допомогою HD Tune, перевірити стан справності за допомогою технології самоконтролю, аналізу та звітності, а також запустити сканування на помилки.

Кажуть, що підтримуваними операційними системами є лише Windows 7, Vista, XP та 2000, але я використовував HD Tune у Windows 10 та Windows 8 без жодних проблем.

DiskCheckup

Переваги:

– Відстежує атрибути SMART для прогнозування збоїв жорсткого диска.

– Можна налаштувати надсилання вам електронних листів про певні

події.

– Добре організований та простий у використанні інтерфейс.

– Невеликий розмір завантаження.

Недоліки:

– Не сканує SCSI або апаратні RAID-масиви.

– Безкоштовно лише для домашнього/особистого використання, не для комерційного/бізнес-користування.

DiskCheckup має працювати з більшістю жорстких дисків. Відображається інформація SMART, така як коефіцієнт помилок читання, час розкрутки, коефіцієнт помилок пошуку та температура, а також короткі та розширені тести диска.

Деталі в розділі SMART можна налаштувати на надсилання електронного листа або відображення сповіщення, коли їхні атрибути перевищують рекомендоване виробником порогове значення.

DiskCheckup має добре працювати з Windows 11, 10, 8, 7, Vista та XP, а також Windows Server 2008 та 2003.

Безкоштовна перевірка диска EASIS

Переваги:

– Результати сканування можуть бути автоматично надіслані вам електронною поштою.

– Виконує сканування поверхні для перевірки на наявність помилок.

– Показує атрибути SMART.

– Перевіряє помилки як на внутрішніх, так і на зовнішніх жорстких дисках.

– Результати сканування показують корисну інформацію.

Недоліки:

– Давно не оновлювався (остання офіційно підтримувана ОС – Windows 7).

– Працює лише на комп'ютерах з ОС Windows.

– Не містить стільки функцій, скільки більшість інших тестерів жорстких дисків.

Безкоштовний EASIS Drive Check – це тестер жорстких дисків, який поєднує в собі дві основні утиліти для тестування: перевірку секторів та зчитувач значень SMART.

Тест SMART відображає понад 40 значень про жорсткий диск, тоді як секторний тест перевіряє поверхню носія на наявність помилок читання.

Звіт про будь-який тест можна прочитати безпосередньо з програми після його завершення, налаштувати для надсилання вам електронною поштою або роздрукувати.

Кажуть, що безкоштовна перевірка диска EASIS працює з Windows 2000 – Windows 7, але я також успішно протестував її у Windows 10 та 8.

Fujitsu Diagnostic Tool

Переваги:

– Одна з найпростіших програм для тестування жорстких дисків.

– Забезпечує дві функції тестування жорсткого диска.

– Працює з Windows, але є також версія для дискети, якщо у вас немає Windows.

Недоліки:

– Тестує лише жорсткі диски Fujitsu.

– Завантажувальна програма запускається лише з дискети (не з диска чи флешки).

– Програму для дискет не так легко встановити та використовувати, як версію для Windows.

– Програма для робочого столу працює лише у Windows.

Fujitsu Diagnostic Tool – це безкоштовний інструмент для тестування жорстких дисків Fujitsu. Він доступний як у версії для Windows, так і у версії для DOS, незалежної від ОС. Однак, завантажувальна версія призначена для дискет – образ, який працює з компакт-диском або USB-накопичувачем, недоступний.

Доступні два тести: «Швидкий тест» (близько трьох хвилин) та «Комплексний тест» (час залежить від розміру жорсткого диска).

Кажуть, що версія для Windows працює з Windows 8, 7, Vista та XP. Вона також може чудово працювати у Windows 11 та Windows 10.

Windows Drive Fitness Test (WinDFT)

Переваги:

– Дуже простий у використанні.

– Є дві функції тестування жорсткого диска.

– Опція дозволяє виконати глибоке тестування для кращих результатів.

– Дозволяє послідовно протестувати кілька дисків.

– Атрибути SMART можна переглянути.

– Також дозволяє стерти дані з жорсткого диска.

Недоліки:

– Не вдається просканувати основний жорсткий диск, на якому інстальовано Windows.

– У програмі немає жодних навчальних посібників, інструкцій чи порад.

– Не вдалося змінити місце збереження файлу LOG.

– Працює лише в операційних системах Windows.

– Більше не отримує оновлень.

Windows Drive Fitness Test – це безкоштовне програмне забезпечення для діагностики жорстких дисків, доступне для використання з більшістю накопичувачів, доступних сьогодні.

Посилання для завантаження нижче встановлює програмне забезпечення Windows Drive Fitness Test в ОС Windows, але ви не можете використовувати програму для сканування диска, на якому встановлено Windows. Сканувати можна лише USB та інші внутрішні жорсткі диски.

Ви можете встановити WinDFT на свій ПК, якщо у вас Windows 11, 10, 8, 7, Vista або XP.

Macrorit Disk Scanner

Переваги:

– Чудові візуальні ефекти, які легко зрозуміти.

– Не обов'язково встановлювати.

– Працює на кількох операційних системах Windows.

Недоліки:

– Виключає функції, доступні лише в платних версіях.

– Безкоштовно лише для особистого/домашнього використання.

– Сканує лише один диск за раз.

– Містить зовнішні посилання, на які ви можете випадково натиснути.

– Нечасті оновлення.

Macrorit Disk Scanner – це проста програма, яка перевіряє жорсткий диск на наявність пошкоджених секторів. Вона проста у використанні, і ви можете почати використовувати її дуже швидко, оскільки вона повністю портативна та не потребує встановлення.

Більша частина цього екрана використовується для візуального відображення прогресу сканування та чіткого відображення наявності пошкоджень.

Працює на Windows 11, 10, 8, 7, Vista та XP. Підтримка Windows Server доступна для платних клієнтів.

Ariolic Disk Scanner

Переваги:

– Перевіряє будь-який жорсткий диск на наявність пошкоджених секторів.

– Показує, на які файли впливають помилки.

– Портативний (не потребує встановлення).

– Дуже чистий інтерфейс, який не відволікає та не заплутує у використанні.

Недоліки:

– Не підтримує HFS (лише файлові системи NTFS та FAT).

– Відображає рекламу після кожного сканування.

Сканер дисків Ariolic схожий на сканер дисків Macrorit тим, що він сканує диск лише для читання на наявність пошкоджених секторів. Він має мінімальний інтерфейс лише з однією кнопкою та легко зрозуміти, чи містять якісь частини диска пошкоджені сектори.

Одна відмінність від Macrorit Disk Scanner полягає в тому, що Ariolic Disk Scanner відображає файли, в яких виникли помилки читання.

Я тестував Ariolic Disk Scanner у Windows 11, 10 та XP, але він також має працювати з іншими версіями Windows. Програма є портативною та важить трохи більше 1 МБ.

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

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

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

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

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

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

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

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

Архітектура платформи .NET Framework

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

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

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 підвищення надійності гібридних систем зберігання інформації SSD/HDD.

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

На практиці застосовуються два основних методи прискорення – кешування даних і їхнє багаторівневе зберігання (tiering). В обох випадках для збільшення продуктивності вводу-виводу використовується концепція «гарячих» даних, але в дійсності це зовсім різні підходи.

При кешуванні один або кілька накопичувачів SSD служать у якості кешу для віртуального пула зберігання, де основне сховище реалізоване на жорстких дисках. SSD у цьому випадку не надають додатковій ємності – це невидима для додатків «прошарок», що збільшує продуктивність вводу-виводу. Інформація завжди передається в основний пул зберігання, однак «гарячі» дані копіюються й у кеш-пам'ять (на SSD). При наступних звертаннях до цим або поруч розміщеним даним замість основного пула зберігання використовується кеш-пам'ять, за рахунок чого й досягається істотний виграш у продуктивності.

При багаторівневому зберіганні дані відповідним чином сортуються й містяться на рівень SSD або HDD (рівнів може бути більше двох): «гарячі» відправляються на флеш-пам'ять, а рідше використовувані – на жорсткі диски.

Багаторівневе зберігання не припускає надмірності даних, тому реалізація RAID у цьому випадку стає більше складною – потрібна покупка додаткових SSD. Саме сортування даних і розподіл їх по рівнях негативно позначаються на продуктивності. Такі системи повинні управляти даними, які з «гарячих» згодом перетворюються в «холодні». Через відсутність надмірності, часто використовувані дані потрібно переміщати в основний пул, як тільки вони стають менш корисними. Ці фонові процеси споживають IOPS і позначаються на швидкості операцій вводу-виводу під час таких переміщень. З найбільшою ефективністю багаторівневе зберігання функціонує в тих випадках, коли

відповідні алгоритми адаптовані до вимог і завдань замовника. Для досягнення ідеальної продуктивності потрібні постійний моніторинг і підстроювання алгоритмів.

На відміну від складного багаторівневого зберігання, кешування на SSD в існуючих СЗД реалізувати простіше. Гібридні системи зберігання з кешуванням на SSD не вимагають додаткового адміністрування, а додаток сприймає таку систему точно так само, як і будь-яку іншу мережну СЗД, тільки працює вона набагато швидше. Реалізація RAID і захист даних у неї аналогічні, і купувати для цього додаткові SSD не буде потрібно.

На SSD містяться копії даних, тому здійснювати їхнє фонове переміщення в основний пул зберігання не прийде. Не буде й пов'язаних із цим витрат, що впливають на продуктивність. Кешування на SSD необхідно, втім, підбудовувати під конкретні корпоративні додатки, але простота системи кешування означає, що адміністрування системи буде значно менш складним, чим у порівнянної СЗД із багаторівневим зберіганням.

Витрати, пов'язані з інсталяцією СЗД із багаторівневим зберіганням і її супроводом, будуть виправданими тільки в дуже великих організаціях, які можуть дозволити собі як установку стійкових модулів SSD для організації виділеного пула флеш-пам'яті, так і збільшення штату системних адміністраторів для керування СЗД. Для більшості компаній, що не мають екстремально великих пулів зберігання, кращим варіантом прискорення систем зберігання буде кешування на SSD.

Будучи більше продуктивними, накопичувачі SSD мають певні обмеження на запис даних, і це потрібно мати на увазі при виборі методу прискорення СЗД. Хоча зберігаються на флеш-накопичувачах дані можна зчитувати нескінченне число раз, їхнього осередку допускають обмежене число циклів запису. Ця проблема ускладнюється необхідністю видаляти весь блок навіть при записі даних меншого обсягу. Для її рішення в сучасних контролерах флеш-пам'яті застосовуються методи розподіленого запису, кешування операцій запису й

фонова «складання сміття». Однак запис на SSD залишається більше складною операцією, чим читання. Занадто часте виконання запису в ті самі осередки може привести до швидкої деградації флеш-пам’яті.

Якщо в клієнтській системі операції запису на SSD можна розподілити таким чином, що кожний окремий блок носія буде перезаписуватися досить рідко, те в гібридної СЗД рівень SSD активно задіється для зберігання «гарячих» даних усього дискового пула. При кешуванні й багаторівневому зберіганні операції з SSD стануть дуже інтенсивними, і переваги алгоритмів запобігання зношування носія будуть зведені на немає. Це означає, що в обох випадках (кешування й багаторівневе зберігання) рівень SSD найкраще задіяти для прискорення операцій читання, а не читання й записи.

У системі з кешуванням на SSD операція вводу-виводу виробляється звичайним образом: спочатку виконуються читання-запис на HDD. Якщо ця операція ініціює кешування, дані також копіюються з HDD на SSD. Тоді при будь-якій наступній операції читання того ж логічного блоку він зчитується безпосередньо з SSD, що збільшує загальну продуктивність і зменшує час відгуку. Рівень SSD відіграє роль невидимого прискорювача вводу-виводу, і при будь-якій відмові SSD дані однаково будуть доступні в основному пулі зберігання, що захищається за допомогою RAID.

Кеш, як і основна ємність зберігання, розбивається на групи секторів рівного розміру. Кожна група називається кеш-блоком, а кожний блок складається з підблоків. Розмір кеш-блоку можна налаштувати під конкретний додаток, наприклад СУБД або Web-сервер. Зчитування даних з HDD і їхній запис в SSD називають наповненням кеш-пам'яті. Ця фонова операція звичайно виконується слідом за основною операцією читання або запису. Оскільки призначення кешу – зберігання часто використовуваних даних, до його наповнення повинна приводити не кожна операція вводу-виводу, а тільки та, для якої граничне значення лічильника виявляється перевищеним. Звичайно лічильники наповнення застосовуються при читанні й при записі.

Таким чином, з кожним блоком основної ємності зберігання асоціюються лічильники читання й запису. Коли додаток зчитує дані з кеш-блоку, значення його лічильника читання збільшується. Якщо дані в кеш-пам'яті відсутні, а значення лічильника читання більше або дорівнює значенню наповнення при читанні, то паралельно з основною операцією читання виконується операція наповнення кеш-пам'яті (дані кешуються). Якщо ж дані вже є в кеш-пам'яті, вони зчитуються з SSD, а операція наповнення не здійснюється. Якщо значення лічильника читання менше граничного значення, воно збільшується, а операція наповнення не виконується. Для операції запису сценарій той же. Докладніше він пояснюється на ілюстраціях на попередньому розвороті.

Що відбувається із умістом кешу після його «розігріву»? Якщо на SSD є вільне місце, кеш продовжує заповнюватися «гарячими» даними. Коли ємність SSD вичерпується, застосовується алгоритм перезапису найменш використовуваних даних (Least Recently Used, LRU), тобто на місце останніх у кеш-пам'яті записуються нові «гарячі» дані.

Якщо обсяг «гарячих» даних перевищує ємність SSD, відсоток зчитувальних з кеш-пам'яті даних зменшується, відповідно, знижується й продуктивність. Крім того, чим менше ємність SSD (і чим більше обсяг гарячих даних), тим інтенсивніше обмін «гарячих» даних. У результаті SSD буде зношуватися швидше.

Фахівці Qsan рекомендують використовувати накопичувачі Intel SSD DC S3500. Так, в SSD ємністю 480 Гбайт наробіток на відмову (MTBF) становить 2 млн ч. Що стосується продуктивності, те типова затримка в цих накопичувачів дорівнює 50 мс, максимальна затримка при читанні – 500 мс (99,9% часу), а продуктивність при довільному читанні блоками по 4 Кбайт досягає 75 тис. IOPS, при записі – 11 тис. IOPS. Це гарний варіант для SSD-кешування.

Кешування при читанні-запису

Операція читання при відсутності даних у кеш-пам'яті відбувається в такий спосіб:

– Додаток подає запит на читання даних.

– Дані зчитуються з HDD.

– Запитані дані вертаються додатку.

– Виконується операція наповнення SSD.

Операція читання при наявності даних у кеш-пам'яті:

– Додаток подає запит на читання даних.

– Дані зчитуються з SSD.

– Запитані дані вертаються додатку.

– При збої SSD дані зчитуються з HDD.

Дії додатка при записі даних:

– Додаток подає запит на запис даних.

– Дані записуються на HDD.

– Додатку вертається статус операції.

– Виконується операція наповнення кеш-пам'яті на SSD.

Налаштування кеш-пам'яті SSD

Щоб додаток використовував кеш-пам'ять на SSD максимально ефективно, її можна налаштувати. Основні параметри – розмір блоку кеш-пам'яті, граничні значення наповнення при читанні й при записі.

Розмір блоку. Великий розмір блоку кеш-пам'яті підходить для додатків, що часто звертаються до сусіднього (по фізичному розташуванню) даним. Це називається високою локальністю звертань. Збільшення розміру блоку також прискорює наповнення кеш-пам'яті на SSD – прискорюється «розігрів» кешу, після якого додатка з високою локальністю звертань будуть демонструвати досить високу продуктивність. Однак збільшення розміру блоку спричиняє генерування надлишкового трафіку вводу-виводу й збільшення часу відгуку, особливо для відсутніх у кеші даних.

Менший розмір блоку гарний для додатків з менш локалізованими даними, тобто коли доступ до даних здійснюється в основному випадковим образом. Кеш-пам'ять на SSD буде «розігріватися» повільніше, але чим більше

блоків, тим більше ймовірність влучення в кеш потрібних даних, особливо даних з низькою локальністю звертань. При невеликих блоках коефіцієнт використання кеш-пам'яті нижче, але менше будуть і супутні втрати, так що при «промаху», коли потрібних даних немає в кеш-пам'яті, продуктивність страждає менше.

Граничне значення наповнення. Поріг наповнення кешу – це число звертань до даних, після якого відповідний блок копіюється в SSD-кеш. При великому значенні кешуються тільки часто використовувані дані й зменшується обмін даних у кеші, але збільшується час «розігріву» кеш-пам'яті й росте ефективність її використання. При меншому значенні кеш-пам'ять розігрівається швидше, але можливо її надлишкове наповнення. Для більшості додатків цілком достатньо граничного значення, рівного 2. Наповнення при записі корисно в тому випадку, коли записувані дані незабаром знову зчитуються. Подібне нерідко трапляється у файлових системах. Інші додатки, наприклад бази даних, не мають такої особливості, тому наповнення при записі для них іноді краще зовсім відключити.

Як можна бачити, збільшення або зменшення кожного параметра має свої позитивні й негативні наслідки. Дуже важливо розуміти «локальність» додатка. Крім того, корисно протестувати систему на реальних навантаженнях і подивитися, при яких параметрах вона показує кращі результати.

3.2 Розробка структурної схеми

У тесті моделювалася типова ситуація вводу-виводу (довільне читання 90% + запис 10%) для визначення виграшу, що дає використання SSD-кешу. При тестуванні застосовувалася система AegisSAN Q500 у наступній конфігурації:

– HDD: Seagate Constellation ES, ST1000NM0011, 1 Тбайт, SATA 6 Гбіт/с (x8);

– SSD: Intel SSD DC 3500, SSDSC2BB480G4, 480 Гбайт, SATA 6 Гбіт/с (x5);

– RAID-група: RAID 5;

– тип вводу-виводу: Database Service (8 Кбайт);

– режим вводу-виводу: блоки по 8 Кбайт.

Час «розігріву» обчислюється по наступній формулі:

$$T = (C \times P) / (I \times S \times D),$$

де T – час «розігріву», I – середня продуктивність в IOPS одного HDD при довільному читанні, S – розмір блоку вводу-виводу, D – число HDD, C – сукупна ємність всіх SSD, P – граничне значення наповнення кеш-пам'яті при читанні або записі. На практиці «розігрів» кешу може зайняти більше часу.

Для даної конфігурації воно складе:

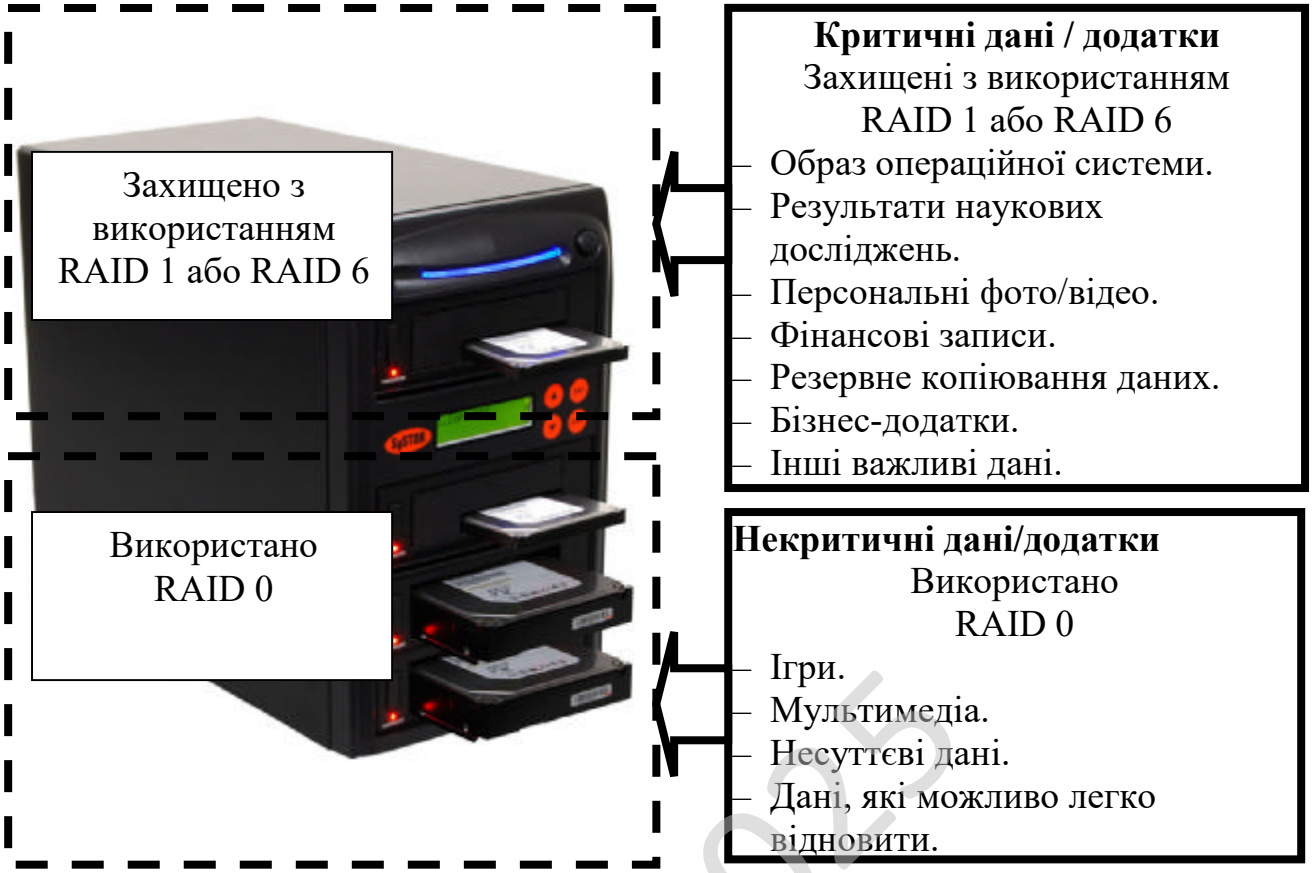
T = (2 Тбайт x 2) / (244 x 8 Кбайт x 8) = 275 036,33 сек = 76,40 год.

Без кешування на SSD середня продуктивність склала 962 IOPS. При включенні кешування вона виросла до 1942 IOPS, тобто поліпшення після «розігріву» кешу виявилося дворазовим – 102%. Відповідно до розрахункової формули час розігріву дорівнює 76,4 год, у тесті після 75 год продуктивність в IOPS досягла максимальної величини й залишалася після цього стабільної.

У концепції прискорення гібридних СЗД реалізується ідея збільшення продуктивності всієї системи за рахунок швидкого доступу до «гарячого» даним. Беручи до уваги витрати на встаткування й адміністрування, можна затверджувати, що в загальному випадку кешування даних на SSD являє собою найкращий спосіб одержання переваг високої продуктивності при використанні систем зберігання із флеш-накопичувачами без втрати надійності зберігання даних.

Для підвищення надійності гібридних систем зберігання інформації SSD/HDD у роботі пропонується використовувати RAID масиви.

Структурна схема розробленої системи зображена на рисунку 3.1.



Гібридна система зберігання інформації SSD/HDD

Рисунок 3.1 – Структурна схема розробленої системи

3.3 Розробка функціональної схеми

Розрізняють кілька основних рівнів RAID-масивів: RAID 0, 1, 2, 3, 4, 5, 6, 7. Також існують комбіновані рівні, такі як RAID 10, 0+1, 30, 50, 53 і т.п. Розглянемо принципи функціонування, достоїнства й недоліки основних рівнів.

RAID 0 – Дисковий масив без відказостійкості (Striped Disk Array without Fault Tolerance)

Дисковий масив без надлишкового зберігання даних. Інформація розбивається на блоки, які одночасно записуються на окремі диски, що забезпечує підвищення продуктивності. Такий спосіб зберігання інформації

ненадійний, оскільки поломка одного диска приводить до втрати всієї інформації, тому рівнем RAID як таким не є.

RAID 0 – дешевий і продуктивний, але ненадійний. За рахунок можливості одночасного вводу/виводу з декількох дисків масиву RAID 0 забезпечує максимальну швидкість передачі даних і максимальну ефективність використання дискового простору, тому що не потрібно місця для зберігання контрольних сум. Реалізація цього рівня дуже проста. RAID 0, як правило, застосовується в тих областях, де потрібна швидка передача великого обсягу даних. Для реалізації масиву потрібно не менше двох вінчестерів.

Переваги:

– найвища продуктивність у додатках, що вимагають інтенсивної обробки запитів вводу/виводу й даних великого обсягу;

– простота реалізації;

– низька вартість;

– максимальна ефективність використання дискового простору – 100%.

Недоліки:

– не є "сьогоденням" RAID'ом, оскільки не підтримує відказостійкість;

– відмова одного диска спричиняє втрату всіх даних масиву.

RAID 1 – Дисковий масив із відзеркалюванням (Mirroring & Duplexing)

Дисковий масив з дублюванням інформації (відзеркалюванням даних). У найпростішому випадку два накопичувачі містять однакову інформацію і є одним логічним диском. При виході з ладу одного диска його функції виконує інший. Для реалізації масиву потрібно не менше двох вінчестерів.

RAID 1 – найпростіший відказостійкий масив.

Переваги:

– простота реалізації;

– простота відновлення масиву у випадку відмови (копіювання).

Недоліки:

– висока вартість – 100-процентна надмірність;

– невисока швидкість передачі даних.

RAID 2 – Відказостійкий дисковий масив з використанням коду Хемминга (Hamming Code ECC)

Схема резервування даних з використанням коду Хеммінга (Hamming code) для корекції помилок. Потік даних розбивається на слова – причому розмір слова відповідає кількості дисків для запису даних. Для кожного слова обчислюється код корекції помилок, що записується на диски, виділені для зберігання контрольної інформації. Їхнє число дорівнює кількості біт у слові контрольної суми.

RAID 2 не одержав комерційного застосування.

Якщо слово складається із чотирьох біт, то під контрольну інформацію приділяється три диски. RAID 2 – один з деяких рівнів, що дозволяють виявляти подвійні помилки й виправляти "на льоту" одиночні. При цьому він є самим надлишковим серед всіх рівнів з контролем парності. Ця схема зберігання даних не одержала комерційного застосування, оскільки погано справляється з великою кількістю запитів.

Переваги:

– досить проста реалізація;

– корекція помилок "на льоту";

– дуже висока швидкість передачі даних;

– при збільшенні кількості дисків накладні витрати зменшуються.

Недоліки:

– низька швидкість обробки запитів;

– висока вартість;

– більша надмірність.

RAID 3 – Відказостійкий дисковий масив з паралельною передачею даних і парністю (Parallel Transfer Disks with Parity)

Відказостійкий масив з паралельним вводом/виводом даних і диском контролю парності. Потік даних розбивається на порції на рівні байт (хоча можливо й на рівні біт) і записується одночасно на всі диски масиву, крім одного. Один диск призначений для зберігання контрольних сум, що обчислюються при записі даних. Поломка кожного з дисків масиву не приведе до втрати інформації.

В RAID 3 інформація розбивається на порції однакового розміру.

Цей рівень має набагато меншу надмірність, чим RAID 2. У RAID 2 більшість дисків, що зберігають контрольну інформацію, потрібні для визначення несправного розряду. Як правило, RAID-контролери можуть одержати дані про помилку за допомогою механізмів відстеження випадкових збоїв. За рахунок розбивки даних на порції RAID 3 має високу продуктивність. Оскільки при кожній операції вводу/виводу виробляється обіг практично до всіх дисків масиву, те одночасна обробка декількох запитів неможлива. Цей рівень підходить для додатків з файлами великого обсягу й малою частотою обігів (в основному це сфера мультимедіа). Використання тільки одного диска для зберігання контрольної інформації пояснює той факт, що коефіцієнт використання дискового простору досить високий (як наслідок цього – відносно низька вартість). Для реалізації масиву потрібно не менше трьох вінчестерів.

Переваги:

– відмова диска мало впливає на швидкість роботи масиву;

– висока швидкість передачі даних;

– високий коефіцієнт використання дискового простору.

Недоліки:

– складність реалізації;

– низька продуктивність при великій інтенсивності запитів даних невеликого обсягу.

RAID 4 – Відказостійкий масив незалежних дисків із загальним диском парності (Independent Data Disks with Shared Parity Disk)

Цей масив дуже схожий на рівень RAID 3. Потік даних розділяється не на рівні байтів, а на рівні блоків інформації, кожний з яких записується на окремий диск. Після запису групи блоків обчислюється контрольна сума, що записується на виділений для цього диск.

В RAID 4 потік даних розділяється на блоки.

В RAID 4 можливо одночасне виконання декількох операцій читання. Цей масив підвищує продуктивність передачі файлів малого обсягу (за рахунок розпаралелювання операції зчитування). Але оскільки при записі повинна змінюватися контрольна сума на виділеному диску, одночасне виконання операцій неможливо (у наявності асимметричність операцій вводу й виводу). Цей рівень має майже всі недоліки RAID 3 і не забезпечує переваги у швидкості при передачі даних великого обсягу. Схема зберігання розроблялася для додатків, у яких дані споконвічно розбиті на невеликі блоки, тому немає необхідності розбивати їх додатково. Ця схема зберігання даних має невисоку вартість, але її реалізація досить складна, як і відновлення даних при збої.

Переваги:

– висока швидкість передачі даних;

– відмова диска мало впливає на швидкість роботи масиву;

– високий коефіцієнт використання дискового простору.

Недоліки:

– досить складна реалізація;

– дуже низька продуктивність при записі даних;

– складне відновлення даних.

RAID 5 – Відказостійкий масив незалежних дисків *з* розподіленою парністю (Independent Data Disks with Distributed Parity Blocks)

Найпоширеніший рівень. Блоки даних і контрольних сум циклічно записуються на всі диски масиву, відсутній виділений диск для зберігання інформації про парність, немає асимметричності конфігурації дисків.

У випадку RAID 5 всі диски масиву мають однаковий розмір – але один з них не бачимий для операційної системи. Наприклад, якщо масив складається з п'яти дисків ємністю 10 Гб кожний, те фактично розмір масиву буде дорівнює 40 Гб – 10 Гб приділяється на контрольні суми. У загальному випадку корисна ємність масиву з N дисків дорівнює сумарної ємності N- 1 диска.

В RAID 5 відсутній виділений диск для зберігання інформації про парність.

Найбільший недолік рівнів RAID від 2-го до 4-го – це наявність окремого диска (або дисків), що зберігає інформацію про парність. Швидкість виконання операцій зчитування досить висока, тому що не вимагає звертання до цього диска. Але при кожній операції запису на ньому змінюється інформація, тому схеми RAID 2-4 не дозволяють проводити паралельні операції запису. RAID 5 не має цього недоліку, тому що контрольні суми записуються на всі диски масиву, що уможливлює виконання декількох операцій читання або записи одночасно. RAID 5 має досить високу швидкість запису/читання й малу надмірність.

Переваги:

– висока швидкість запису даних;

– досить висока швидкість читання даних;

– висока продуктивність при великій інтенсивності запитів читання/запису даних;

– високий коефіцієнт використання дискового простору.

Недоліки:

– низька швидкість читання/запису даних малого обсягу при одиничних запитах;

– досить складна реалізація;

– складне відновлення даних.

RAID 6 – Відказостійкий масив незалежних дисків із двома незалежними розподіленими схемами парності (Independent Data Disks with Two Independent Distributed Parity Schemes)

RAID 6 – це відказостійкий масив незалежних дисків з розподілом контрольних сум, обчислених двома незалежними способами. Цей рівень багато в чому схожий з RAID 5. Тільки в ньому використовується не одна, а дві незалежні схеми контролю парності, що дозволяє зберігати працездатність системи при одночасному виході з ладу двох накопичувачів. Для обчислення контрольних сум в RAID 6 використовується алгоритм, побудований на основі коду Ріда-Соломона (Reed-Solomon).

RAID 6 використовує дві незалежні схеми контролю парності.

Цей рівень має дуже високу відказостійкість, більшу швидкість зчитування (дані зберігаються блоками, немає виділених дисків для зберігання контрольних сум). У той же час через великий обсяг контрольної інформації RAID 6 має низьку швидкість запису. Він дуже складний у реалізації, характеризується низьким коефіцієнтом використання дискового простору: для масиву з п'яти дисків він становить усього 60%, але з ростом числа дисків ситуація виправляється.

RAID 6 по багатьом характеристикам програє іншим рівням, тому на сьогодні не одержав комерційного застосування.

Переваги:

– висока відказостійкість;

– досить висока швидкість обробки запитів;

Недоліки:

– низька швидкість читання/запису даних малого обсягу при одиничних запитах;

– дуже складна реалізація;

– складне відновлення даних;

– низька швидкість запису даних.

RAID 7 – Відказостійкий масив, оптимізований для підвищення продуктивності (Optimized Asynchrony for High I/O Rates as well as High Data Transfer Rates)

На відміну від інших рівнів, RAID 7 не є відкритим індустріальним стандартом – це зареєстрована торговельна марка компанії Storage Computer Corporation. Масив ґрунтується на концепціях, використаних у третьому й четвертому рівнях. Додалася можливість кешування даних. До складу RAID 7 входить контролер з убудованим мікропроцесором під керуванням операційної системи реального часу (real-time OS). Вона дозволяє обробляти всі запити на передачу даних асинхронно й незалежно.

RAID 7 – зареєстрована торговельна марка компанії Storage Computer Corporation.

Блок обчислення контрольних сум інтегрований із блоком буферізації; для зберігання інформації про парність використовується окремий диск, що може бути розміщений на будь-якому каналі. RAID 7 має високу швидкість передачі даних і обробки запитів, гарну масштабованість. Самим більшим недоліком цього рівня є вартість його реалізації.

Переваги:

– дуже висока швидкість передачі даних і висока швидкість обробки запитів (в 1,5...6 разів вище інших стандартних рівнів RAID);

– гарна масштабованість;

– значно зросла (завдяки наявності кешу) швидкість читання даних невеликого обсягу;

– відсутність необхідності в додатковій передачі даних для обчислення парності.

Недоліки:

– власність однієї компанії;

– складність реалізації;

– дуже висока вартість на одиниці об'єму;

– не може обслуговуватися користувачем;

– необхідність використання блоку безперебійного живлення для запобігання втрати даних з кеш-пам'яті;

– короткий гарантійний строк.

RAID 10

RAID 10 – відказостійкий масив з дублюванням і паралельною обробкою. Ця архітектура виявляє собою масив типу RAID 0, сегментами якого є масиви RAID 1. Він поєднує в собі високу відказостійкість і продуктивність.

Комбіновані рівні

Крім базових рівнів RAID 0 – RAID 5, описаних у стандарті, існують комбіновані рівні RAID 1+0, RAID 3+0, RAID 5+0, RAID 1+5, які різні виробники інтерпретують кожний по-своєму.

– RAID 1+0 – це сполучення дзеркалювання й чергування.

Нинішні контролери використовують цей режим за замовчуванням для RAID 1. Тобто, 1 диск основний, 2-й диск – дзеркало, причому читання виробляється з них по черзі, як для RAID 0. Властиво, зараз можна вважати що RAID 1 і RAID 1+0 – це просто різна назва того самого методу апаратного дзеркалювання дисків. Але не варто забувати, що повноцінний RAID 1+0 повинен містити як мінімум 4 диски.

– RAID 5+0 – це чергування томів 5-го рівня. RAID 1+5 – RAID 5 із дзеркальованою парою.

Комбіновані рівні успадковують як переваги, так і недоліки своїх «батьків»: поява чергування в рівні RAID 5+0 анітрошки не додає йому надійності, але зате позитивно відбивається на продуктивності. Рівень RAID 1+5, напевно, дуже надійний, але не найшвидший і, до того ж, украй неекономічний: корисна ємність тому менше половини сумарної ємності дисків.

Варто відзначити, що кількість жорстких дисків у комбінованих масивах також зміниться. Наприклад для RAID 5+0 використовують 6 або 8 жорстких дисків, для RAID 1+0 – 4, 6 або 8.

Matrix RAID

Matrix RAID – це технологія, реалізована фірмою Intel у своїх чипсетах починаючи з ICH6R. Строго говорячи, ця технологія не є новим рівнем RAID (її аналог існує в апаратних RAID-контролерах високого рівня), вона дозволяє, використовуючи невелику кількість дисків організувати одночасно один або кілька масивів рівня RAID 1, RAID 0 і RAID5. Це дозволяє за порівняно невеликі гроші забезпечити для одних даних підвищену надійність, а для інших високу швидкість доступу.

Програмний RAID

Для реалізації RAID можна застосовувати не тільки апаратні засоби, але й повністю програмні компоненти (драйвери). Наприклад, у системах на ядрі Linux існують спеціальні модулі ядра, а управляти RAID-пристроями в GNU/Linux можна за допомогою утиліти mdadm. Програмний RAID має свої достоїнства й недоліки. З одного боку, він нічого не коштує (на відміну від апаратних RAID-контролерів, ціна яких від $250). З іншого боку, програмний RAID використовує ресурси центрального процесора, і в моменти пікового навантаження на дискову систему процесор може значну частину потужності витрачати на обслуговування RAID-пристроїв.

Ядро GNU/Linux 2.6.28 (останнє з вийшли в 2008 році) підтримує програмні RAID наступних рівнів: 0, 1, 4, 5, 6, 10. Реалізація дозволяє створювати RAID на окремих розділах дисків, що аналогічно описаному вище Matrix RAID. Завантаження підтримується (grub, lilo) тільки з дисків з RAID 1.

ОС Windows 2007/2008/2010 підтримує програмний RAID 0, RAID 1 і RAID 5. Більш точно, Windows 10/11 Pro підтримує RAID 0. Підтримка RAID 1 і RAID 5 заблокована розроблювачами, але, проте, може бути включена, шляхом

редагування системних бінарних файлів ОС. Windows Server 2003 – 0, 1 і 5. Windows 10/11 Home RAID не підтримує.

В ОС FreeBSD є кілька реалізацій програмного RAID. Так, atacontrol, може як повністю будувати програмний RAID, так і може підтримувати напівапаратний RAID на таких чипах як ICH5R. В FreeBSD, починаючи з версії 5.0, дискова підсистема управляється убудованим у ядро механізмом GEOM. GEOM надає модульну дискову структуру, завдяки якій народилися такі модулі як gstripe (RAID 0), gmirror (RAID 1), gRAID3 (RAID 3), gconcat (об'єднання декількох дисків у єдиний дисковий розділ). Так само існують застарілі класи ccd (RAID 0, RAID 1) і gvinum (менеджер логічних томів vinum). Починаючи з FreeBSD 7.0 підтримується файлова система ZFS у якій можна збирати наступні рівні RAID: 0, 1, 5, 6, а так само комбинуємі рівні.

OpenSolaris і Solaris 10 використовують Solaris Volume Manager, що підтримує RAID-0, RAID-1, RAID-5 і будь-які їхні комбінації як 1+0. Підтримка RAID-6 здійснюється у файловій системі ZFS.

Система підвищення надійності збереження інформації на SSD дисках з використанням технології RAID, яка розроблена у результаті виконання дипломного проектування, будується на використанні технології Intel Intel® SATA Solid-State Drive, з використанням RAID 0 та RAID 1 або RAID 6, в залежності від кількості дисків у RAID-масиві. Якщо диска два, то використовується RAID 1, якщож їх більше то використовується RAID 6.

RAID 0 дозволяє розбивати інформацію на блоки, які одночасно записуються на окремі диски, що забезпечує підвищення продуктивності. Такий спосіб зберігання інформації ненадійний, оскільки поломка одного диска приводить до втрати всієї інформації.

При використанні RAID 1 два накопичувачі містять однакову інформацію і є одним логічним диском. При виході з ладу одного диска його функції виконує інший. Для реалізації масиву потрібно не менше двох вінчестерів.

RAID 6 дозволяє сформувати відказостійкий масив незалежних дисків з розподілом контрольних сум, обчислених двома незалежними способами. В ньому використовується дві незалежні схеми контролю парності, що дозволяє зберігати працездатність системи при одночасному виході з ладу двох накопичувачів. Для обчислення контрольних сум в RAID 6 використовується алгоритм, побудований на основі коду Ріда-Соломона (Reed-Solomon).

Intel також анонсувала новий, більше гнучкий контролер накопичувачів за назвою Intel® SATA Solid-State Drive. Він може підтримувати декілька одночасно працюючих масивів RAID на тому самому наборі приводів. Нагадаємо, що масив RAID використовує одночасно кілька приводів, що дозволяє підвищити продуктивність підсистеми зберігання, одночасно забезпечуючи захист від збою привода.

Драйвер Intel® SATA Solid-State Drive підтримує різні режими міграції, включаючи перехід з одного привода на масив RAID або додавання привода до існуючого масиву RAID.

Контролер SATA підтримує "рідну" чергу команд, а також швидкість 300 Мбайт/с для кожного порту SATA. Втім, така швидкість навряд чи дає яку-небудь перевагу, оскільки швидкість читання із пластин не перевищує 80 Мбайт/с, а швидкість читання з кеша вінчестера прямо мало впливає на продуктивність.

Контролер SATA у південному мосту ICH7 обзавівся підтримкою Advanced Host Controller Interface (ICHI). Специфікація 1.1 описує інтерфейс на рівні регістрів і допомагає стандартизації контролерів SATA-II. Попередні реалізації працювали на власних схемах, тому подібний крок досить важливий для ефективного використання черги команд. Але поки ще занадто рано оцінювати ступінь важливості.

Реально програмне забезпечення створює віртуальні диски на яких імітується RAID-масив. Для цього створюється як мінімум 2 віртуальні диски, які дозволяють гарантовано зберігати інформацію за допомогою RAID 0 та RAID 1, якщо дисків створюється більше ніж 2 то гарантовано зберігається інформація за

допомогою RAID 0 та RAID 6.

Із двома портами Serial ATA контролер Matrix RAID можна настроїти на два масиви RAID 0 або RAID 1. У чипсетів 945P і 955X підтримуються чотири порти SATA, так що й опцій тут більше. Із трьома встановленими жорсткими дисками SATA південний міст ICH 7-R підтримує ще й масив RAID 6, а із чотирма вінчестерами – RAID 6 і RAID 10. RAID 0 може працювати з кількістю жорстких дисків до чотирьох, у той час як RAID 1 доступний тільки по парах.

Досить цікавою можна назвати можливість сполучення швидкого масиву (приміром, RAID 0) з надлишковим масивом (типу RAID 1 або 6). Наприклад, при використанні двох приводів на RAID 0 можна встановити операційну систему й програми. На другому масиві RAID 1 можна зберігати документи, важливі файли або навіть образ операційної системи для відновлення.

Із чотирма дисками можливо ще більше варіантів. Скажемо, сполучення RAID 0 і RAID 6. Але варто пам'ятати, що три жорсткі диски є мінімально необхідною умовою для RAID 6, що гарантує схоронність даних у випадку виходу з ладу одного жорсткого диска.

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

Блок вибору типу RAID масиву дозволяє автоматично, в залежності від заданих параметрів у попередніх блоках, визначити, який саме тип RAID-масиву гібридних систем зберігання інформації SSD/HDD потрібен для зберігання тієї, або іншої інформації.

Система складається з наступних блоків:

– Блок розподілу інформації за критерієм критичності/важливості визначає який саме вид RAID-масиву гібридних систем зберігання інформації SSD/HDD необхідно буде використовувати. Якщо інформація не є критичною то буде використовуватися RAID 0. Якщо інформація є важливою, то в залежності від ступеня важливості, та вимог до швидкодії, буде використовуватися або RAID 1, який є більш швидким, та менш надійним, або RAID 6, який є дуже

надійним але менш швидким.

– Блок вибору кількості дисків – призначений для визначення кількості дисків з яких буде складатися RAID-масив гібридних систем зберігання інформації SSD/HDD. Якщо дисків буде 2 то буде використовуватися технологія RAID 0 та RAID 1. Якщо дисків буде більше 2 то буде використовуватися технологія RAID 0 та RAID 6.

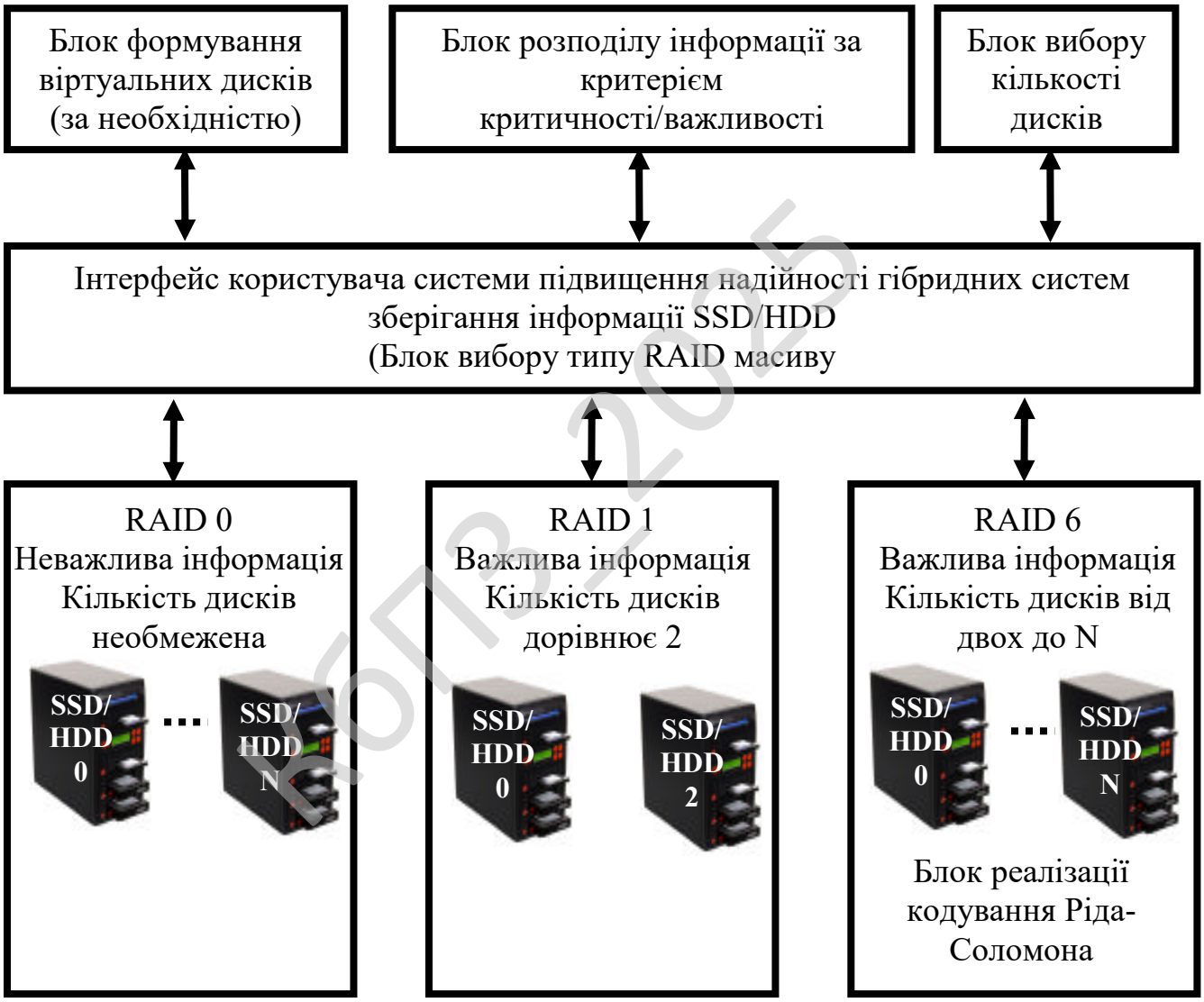


Рисунок 3.2 – Функціональна схема системи

– Блок формування віртуальних дисків використовується у разі коли диск тільки один. Тоді на цьому диску формуються декілька віртуальних дисків, з

якими відбуваються дії аналогічні як ті, що відбуваються над фізичними дисками. Тобто RAID-масив формується з віртуальних дисків.

Блоки RAID 0, RAID 1 та RAID 6 реалізують ту, або іншу технологію.

У блоці RAID 6 використовується кодек Ріда-Соломона. Нижче надамо опис цього кодеку.

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

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

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

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

Поліном, що породжує

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* $g_r = 1$.

Теорема 3.1

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

Теорема 3.2

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

двочлена $x^n - 1$.

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

Матриця, що породжує

Поліноми $g(x), xg(x), x^2g(x), \ldots, x^{k-1}g(x)$ лінійно незалежні, інакше $m(x)g(x) = 0$ при ненульовому $m(x)$, що неможливо.

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

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

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 $H = \begin{bmatrix} 1 & x & x^2 & \ldots & x^{n-2} & x^{n-1} \end{bmatrix} \mod g(x)$.

Тоді:

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

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 $GF(q)$, корінням якого є $d - 1$ підряд, що йдуть ступенів, $\alpha^{i_0}, \alpha^{i_0+1}, ..., \alpha^{i_0+d-1}$ елемента

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

Довжина отриманого коду n, мінімальна відстань d (мінімальна відстань d лінійного коду є мінімальним із всіх відстаней Хеммінга всіх пар кодових слів). Код містить $r = d - 1 = \deg(g(x))$ перевірочний символ, де deg() позначає ступінь полінома; число інформаційних символів $k = n - r = n - d + 1$. У такий спосіб $d = n - k - 1$ і код Ріда-Соломона є *роздільним кодом з максимальною відстанню* (є оптимальним у змісті границі Синглтона).

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

Властивості

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

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

Виправлення багаторазових помилок

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

виправляти будь-яку комбінацію помилок, зосереджену в t або меншому числі блоків з m символів.

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

Практична реалізація

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

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

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

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

Кодування

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

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

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

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

Кодер будується зі регістрів зсуву, суматорів і перемножувачів. Регістр зсуву складається з комірок пам'яті, у кожній з яких перебуває один елемент поля Галуа.

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

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

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

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

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

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

На структурній схемі кодера (рисунок 3.2) символу «С» відповідають дві

константи $GF(n)$.

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

У результаті кодер з урахуванням схеми синхронізації (на рисунку не показана) займає 255 LE (Logic Element – логічний елемент) і 3 EAB, що дозволяє розмістити його в IMC EPF10K10. Після оптимізації розміщення схеми на кристалі FPGA швидкодія схеми досяглася 11,57 Мгц (частота надходження байт даних, далі – байтова частота).

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

Декодування

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

– Обчислює синдром помилки.

– Будує поліном помилки.

– Знаходить корінь даного полінома.

– Визначає характер помилки.

– Виправляє помилки.

Обчислення синдрому помилки

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

Побудова полінома помилки

Обчислений синдром помилки не вказує на положення помилок. Ступінь полінома синдрому дорівнює $2t$, 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-Мессі, 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-Мессі. 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 корня

На цьому етапі шукаються коріння полінома помилки, що визначають положення перекручених символів у кодовому слові. Реалізується за допомогою процедури Ченя, рівносильній повному перебору. У поліном помилок послідовно

підставляються всі можливі значення, коли поліном звертається в нуль – коріння знайдені.

Визначення характеру помилки і її виправлення

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

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

3.4 Розробка діаграми процесів

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

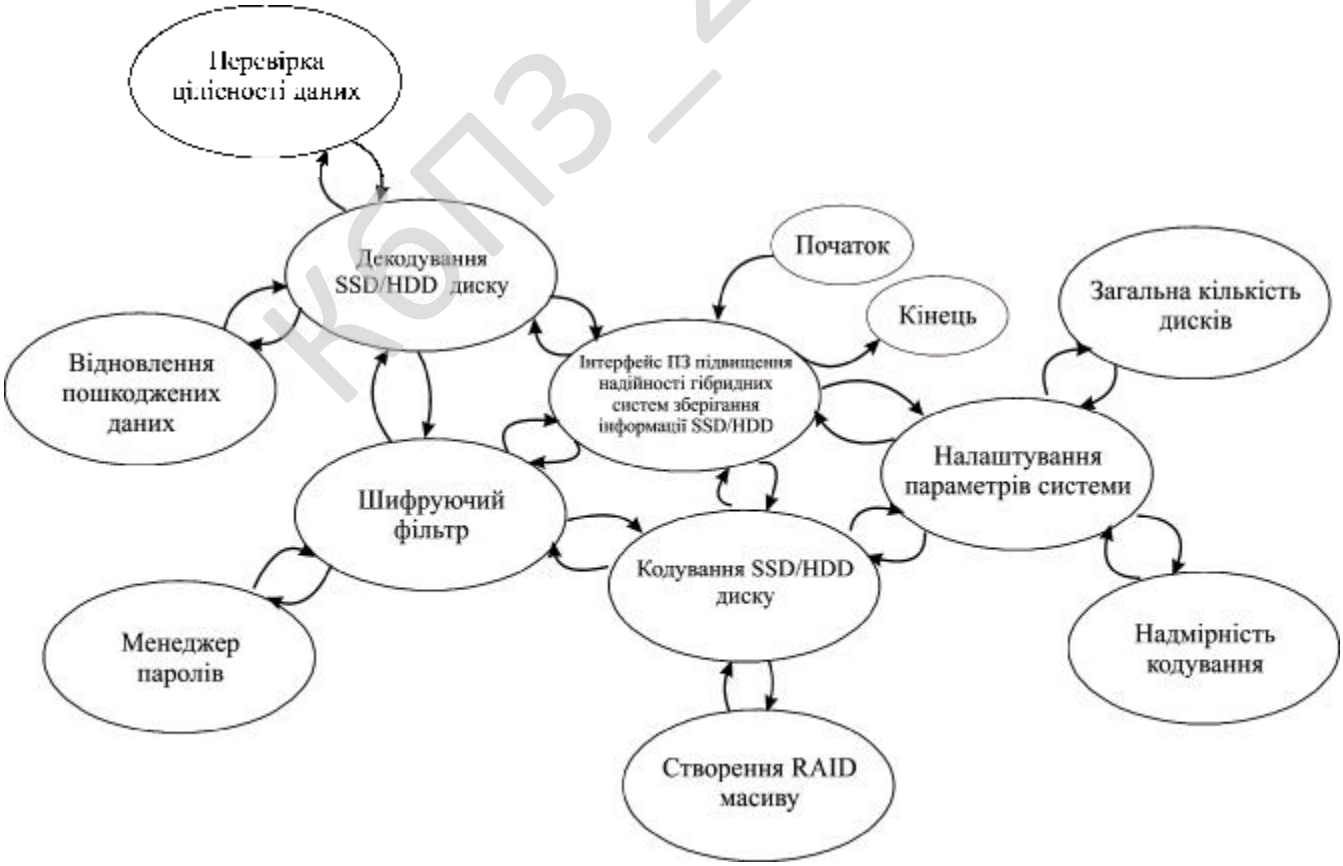


Рисунок 3.3 – Діаграма взаємодії процесів

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

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

Діаграми потоків даних містять чотири типи елементів:

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

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Перед розглядом блок схем необхідно розглянути опис алгоритмів функціонування системи.

Нижче наведемо частину кода, використаного кодера Ріда-Соломона. Кодуємі дані передаються через масив data[i], де i = 0...(k – 1), а згенеровані символи парності заносяться в масив b[0]...b[2t – 1]. Вихідні й результуючі дані повинні бути представлені в поліноміальної формі (тобто у звичайній формі машинного подання даних).

Кодування виробляється з використанням сдвигового feedback-регістра,заповненого відповідними елементами масиву g[] з породженим поліномом усередині, процедура генерації якого вже обговорювалася в опередній главі. Згенероване кодове слово описується наступною формулою:

$$C(x) = data(x) * x \wedge (n - k) + b(x),$$

де ^ означає зведення в ступінь.

```
B_rs()
{
       int i, j;
       int feedback;
// ініціалізація поля біт парності нулями
       for (i = 0; i < n - k; i++) b[i] = 0;
// обробляємо всі символи
// вихідних даних праворуч ліворуч
       for (i = k - 1; i >= 0; i-i--)
       {
// готовимо (data[i] + b[n - k - 1]) до множення на g[i]
// тобто складаємо черговий "захоплений" символ вихідних
// даних з молодшим символом біт парності (відповідного
```

```
// "регістру" b2 t-1, і переводимо його в
// індексну форму, зберігаючи результат у регістрі feedback;
// як ми вже говорили, сума двох індексів є добуток  поліномів
              feedback = index_of[data[i] ^ b[n - k - 1]];
 // є ще символи для обробки?
              if (feedback != -1)
              {
 // здійснюємо зрушення ланцюга bx-регістрів
                     for (j = n - k - 1; j > 0; j-j--)
// якщо поточний коефіцієнт g - це дійсний
// (тобто ненульовий коефіцієнт, те
// множимо feedback на відповідний g-коефіцієнт
// і складаємо його з наступних елементів ланцюжка
if (g[j] != -1) b[j] = b[j - 1] ^ alpha_to[(g[j] + feedback) % n];
                     else
// якщо поточний коефіцієнт g - це нульовий коефіцієнт,
// виконуємо одне лише зрушення без множення, переміщаючи
// символ з одного m-регістра в інший
                           b[j] = b[j - 1];
// закільцьовуємо вихідний символ у крайній лівий b 0-регістр
b[0] = alpha_to[(g[0] + feedback) % n];
              }
              else
              {
// розподіл завершений,
// здійснюємо останнє зрушення регістра,
// на виході регістра буде частка, що губиться,
// а в самому регістрі - шуканий залишок
for (j = n - k - 1; j > 0; j-j--) b[j] = b[j - 1]; b[0] = 0;
              }
       }
}
```

Розглянувши алгоритм та програмну реалізацію процесу кодування перейдемо до процесу декодування закодованого тексту.

Декодування кодів Ріда-Соломона являє собою досить складне завдання, рішення якого виливається в громіздкий, заплутаний і надзвичайно ненаглядний програмний код, що вимагає від розроблювача великих знань у багатьох областях вищої математики.

Типова схема декодування, що одержала назву авторегрессионого

спектрального методу декодування, складається з наступних кроків:

– Обчислення синдрому помилки (синдромный декодер).

– Побудови полінома помилки, здійснювана або за допомогою високоефективного, але складно реалізованого алгоритму Берлекемпа-Мессі, або за допомогою простого, але гальмового Евклідового алгоритму.

– Знаходження корінь даного полінома, що звичайно вирішується лобовим перебором.

– Визначення характеру помилки, що зводиться до побудови бітової маски, що обчислюється на основі обігу алгоритму Форни або будь-якого іншого алгоритму обігу матриці.

– Нарешті, виправлення помилкових символів, шляхом накладення бітової маски на інформаційне слово й послідовне інвертування всіх перекручених біт через операцію XOR.

Наведемо вихідний текст підпрограми декодера Ріда-Соломона. Процедура декодування кодів Ріда-Соломона складається з декількох кроків: спочатку ми обчислюємо 2 t-символьний синдром шляхом постановки alpha**i в recd(x), де recd – отримане кодове слово, попередньо переведене в індексну форму.

По факту обчислення recd(x) ми записуємо черговий символ синдрому в s[i], де i приймає значення від 1 до 2t, залишаючи s[0] рівним нулю. Потім, використовуючи ітеративний алгоритм Берлекемпа (Berlekamp), ми знаходимо поліном локатора помилки – elp[i].

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

Якщо кількість певних позицій перекручених символів менше ступеня elp, перекручуванню піддалося більш ніж t символів і ми не можемо відновити їх. У всіх інших випадках відновлення оригінального вмісту перекручених символів

цілком можливо.

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

```
decode_rs()
{
       int i, j, u, q;
       int s[n – k + 1]; // поліном синдрому помилки
       int elp[n – k + 2][n – k]; // поліном локатора помилки лямда
       int d[n – k + 2];
       int l[n – k + 2];
       int u_lu[n – k + 2],
     int count = 0, syn_error = 0, root[t], loc[t], z[t + 1], err[n], reg[t + 1];
// переводимо отримане кодове слово в індексну форму
// для спрощення обчислень
       for (i = 0; i < n; i++) recd[i] = index_of[recd[i]];
// обчислюємо синдром
//--------------------------------
       for (i = 1; i <= n – k; i++)
       {
              s[i] = 0; // ініціалізація s-регістра
// на його вхід за замовчуванням надходить нуль
// виконуємо s[i] += recd[j] * ij
// тобто беремо черговий символ декодуємих даних,
// множимо його на порядковий номер даного символу,
// помножений на номер чергового оберту й складаємо
// отриманий результат із умістом s-регістра;
// по факті вичерпання всіх декодуємих символ ми
// повторюємо весь цикл обчислень знову – по одному
// разу для кожного символу парності
              for (j = 0; j < n; j++) if (recd[j] != -1) s[i] ^= alpha_to[(recd[j]
+ i * j) % n];
              if (s[i] != 0) syn_error = 1; // якщо синдром не дорівнює нулю,
зводимо прапор помилки
// перетворимо синдром з поліноміальної форми в індексну
              s[i] = index_of[s[i]];
       }
// корекція помилок
//-----------------------
       if (syn_error)    // якщо є помилки, намагаємося їх скорегувати
       {
 // обчислення полінома локатора ламбла
```

```
//---------------------------
// обчислюємо поліном локатора помилки через ітеративний алгоритм
// Берлекемпа. Випливаючи термінології Lin and Costello (див. "Error
// Control Coding: Fundamentals and Applications" Prentice Hall 1983
// ISBN 013283796) d[u] являє собою m ("мю"), що виражає

// розбіжність (discrepancy), де u = m + 1 і m є номер кроку
// з діапазону від -1 до 2t. У Блейхута та ж сама величина
// позначається D(x) ("дельта") і називається нев'язання.
// l[u] являє собою ступінь elp для даного кроку ітерації,
// u_l[u] являє собою різницю між номером кроку й ступенем elp
// ініціалізація елементи таблиці
              d[0] = 0; // індексна форма
              d[1] = s[1]; // індексна форма
              elp[0][0] = 0; // індексна форма
              elp[1][0] = 1; // поліноміальна форма
              for (i = 1; i < n - k; i++)
              {
                     elp[0][i] = -1; // індексна форма
                     elp[1][i] = 0; // поліноміальна форма
              }
              l[0] = 0; l[1] = 0; u_lu[0] = -1; u_lu[1] = 0; u = 0;
              do
              {
                     u++;
                     if (d[u] == -1)
                     {
                            l[u + 1] = l[u];
                            for (i = 0; i <= l[u]; i++)
                            {
                                   elp[u + 1][i] = elp[u][i];
                                   elp[u][i] = index_of[elp[u][i]];
                            }
                     }
                            else
                     {
// пошук слів з найбільшим u_lu[q], таких що d[q] != 0
                            q = u - 1;
                            while ((d[q] == -1) && (q > 0)) q-q--;
// знайдений перший ненульовий d[q]
                            if (q > 0)
                            {
```

```
                                j = q;
                                do
                        {
                                j-j-- ;
                                if ((d[j] != -1) && (u_lu[q] < u_lu[j]))
                                q = j;
                        } while (j > 0);
                };
 // як тільки ми знайдемо q, такий що d[u] !=0
// і u_lu[q] є максимум
// запишемо ступінь нового elp полінома
                if (l[u] > l[q] + u - q) l[u + 1] = l[u]; else l[u + 1] =
l[q] + u - q;
// формуємо новий elp(x)
                for (i = 0; i < n - k; i++) elp[u + 1][i] = 0;
                for (i = 0; i <= l[q]; i++)
                        if (elp[q][i] != -1)
                                elp[u + 1][i + u - q] = alpha_to[(d[u] + n -
 d[q] + elp[q][i]) % n];
                for (i = 0; i <= l[u]; i++)
                {
                        elp[u + 1][i] ^= elp[u][i];
// перетворимо старий elp
// в індексну форму
                        elp[u][i] = index_of[elp[u][i]];
                }
        }
        u_lu[u + 1] = u - l[u + 1];
 // формуємо (u + 1)'ю нев'язання
//--------------------------
        if (u < n - k)
// на останній ітерації розбіжність
        {
// не було виявлено
                if (s[u + 1] != -1) d[u + 1] = alpha_to[s[u + 1]]; else d[u +
1] = 0;
                for (i = 1; i <= l[u + 1]; i++)
if ((s[u + 1 - i] != -1) && (elp[u + 1][i] != 0))
d[u + 1] ^= alpha_to[(s[u + 1 - i] + index_of[elp[u + 1][i]]) % n];
// переводимо d[u + 1] в індексну форму
                d[u + 1] = index_of[d[u + 1]];
        }
```

```
        } while ((u < n - k) && (l[u + 1] <= t));
// розрахунок локатора завершений
//------------------------
        u++ ;
        if (l[u] <= t)
        {
// корекція помилок можлива
// переводимо elp в індексну форму
                for (i = 0; i <= l[u]; i++) elp[u][i] = index_of[elp[u][i]];
// знаходження корінь полінома локатора помилки
//--------------------------
                for (i = 1; i <= l[u]; i++) reg[i] = elp[u][i]; count = 0;
                for (i = 1; i <= n; i++)
                {
                        q = 1 ;
                        for (j = 1; j <= l[u]; j++)
                        if (reg[j] != -1)
                        {
                                reg[j] = (reg[j] + j) % n;
                                q ^= alpha_to[reg[j]];
                        }
                        if (!q)
                        {
// записуємо корінь і індекс позиції помилки
                                root[count] = i;
                                loc[count] = n - i;
                                count++;
                        }
                }
                if (count == l[u])
                {
// немає корінь - ступінь elp < t помилок
// формуємо поліном z(x)
                        for (i = 1; i <= l[u]; i++)
// Z[0] завжди дорівнює 1
                        {
                                if ((s[i] != -1) && (elp[u][i] != -1))
                                        z[i] = alpha_to[s[i]] ^ alpha_to[elp[u][i]];
                                else
                                        if ((s[i]! =- 1) && (elp[u][i] == -1))
                                                z[i] = alpha_to[s[i]];
                                        else
```

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

```
        {
// переводимо recd[] у поліноміальну форму
                for (i = 0; i < n; i++)
                        if (recd[i] != -1) recd[i] = alpha_to[recd[i]];
                else
                        recd[i] = 0;
// виводимо інформаційне слово як є
        }
                else
// ступінь elp > t, рішення неможливо
        {
// переводимо recd[] у поліноміальну форму
        for (i = 0; i < n; i++)
                if (recd[i] != -1)
                        recd[i] = alpha_to[recd[i]];
                else
                        recd[i] = 0 ;
// виводимо інформаційне слово як є
        }
                else
// помилок не виявлене
        for (i = 0; i < n; i++) if (recd[i] != -1) recd[i] = alpha_to[recd[i]];
else recd[i] = 0;
}
```

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.

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

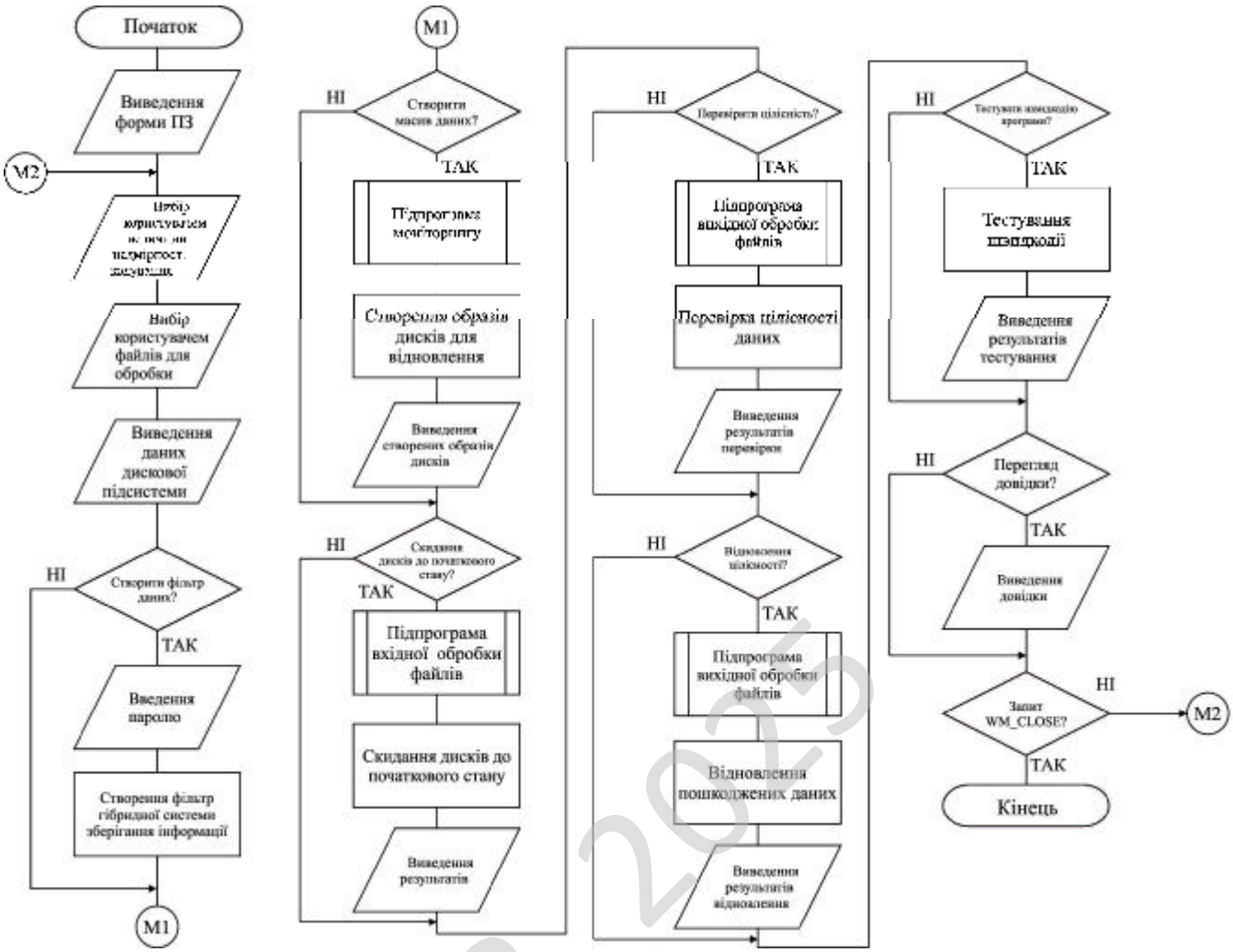


Рисунок 4.1 – Блок-схема основної програми

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю підвищення надійності гібридних систем зберігання інформації SSD/HDD.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення.

UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю.

UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

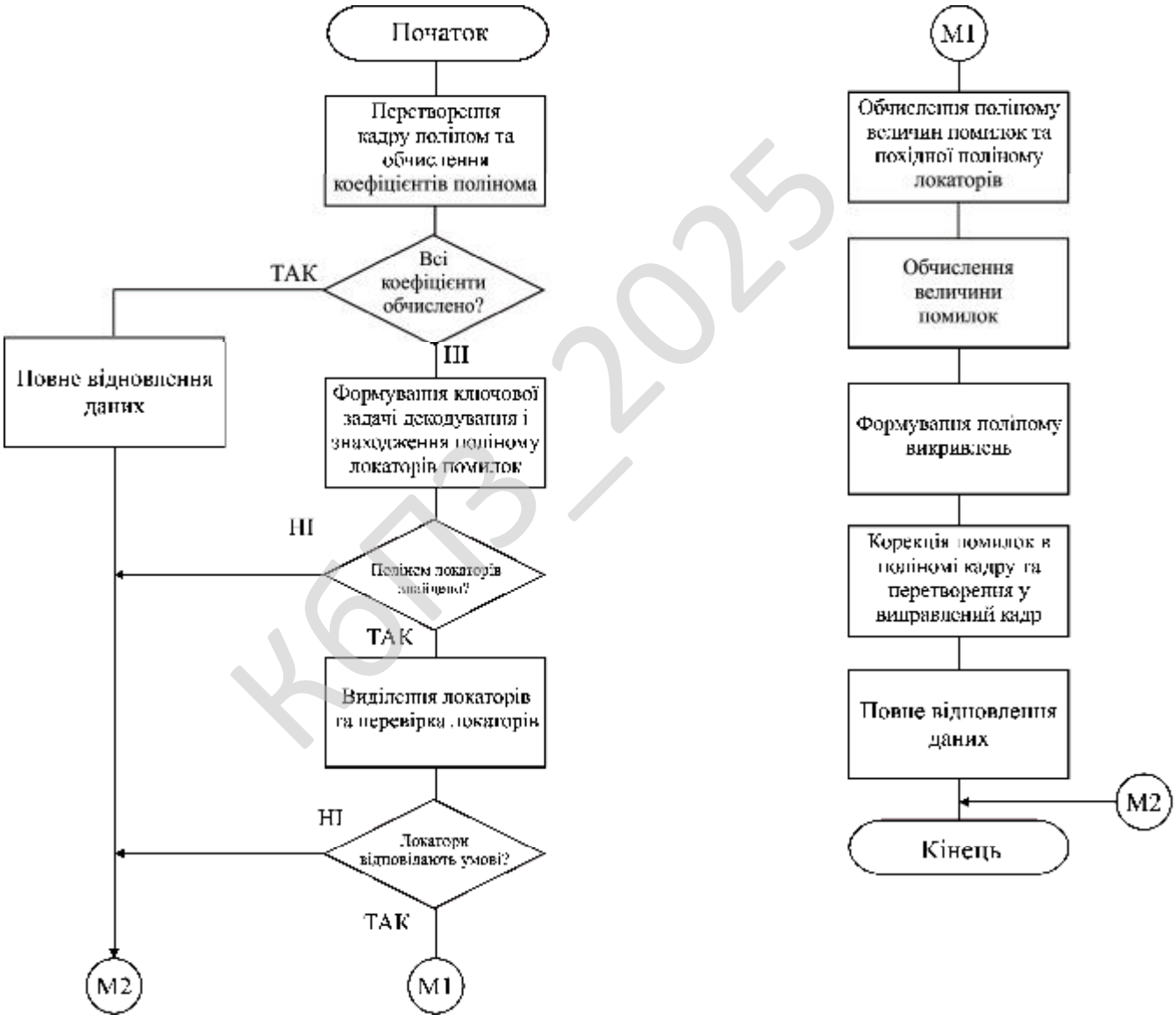


Рисунок 4.2 – Блок-схема роботи підпрограми

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

– Керівникам проектів, які керують розподілом завдань і контролем за проектом.

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

– Бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії.

– Програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

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

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

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

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

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

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

прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

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

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

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

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

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

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

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

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

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

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

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

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

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

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

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

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

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин). Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу. Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням. Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

4.2 Захист розробленого програмного забезпечення

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

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

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

Перевагами шифру є його простота в реалізації, невеликий розмір коду й досить висока швидкість виконання, а також можливість оптимізації виконання на стандартних 32-бітних процесорах, так як в якості основних операцій використовуються операції виключна «АБО» (XOR), побітового зсуву й додавання за модулем 2^{32}. Оскільки алгоритм не використовує таблиць підстановки і раундова функція досить проста, алгоритму потрібно не менше 16 циклів (32 раундів) для досягнення ефективної дифузії, хоча повна дифузія досягається вже через 6 циклів (12 раундів).

Алгоритм має відмінну стійкість до лінійного криптоаналізу і досить гарну до диференціального криптоаналізу. Головним недоліком цього алгоритму

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

Опис алгоритму

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

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

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

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

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

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

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

Де

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

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

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

Константа δ була виведена з Золотого перерізу:

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

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

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

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

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання бакалаврської дипломної роботи. Розроблене програмне забезпечення підвищення надійності гібридних систем зберігання інформації SSD/HDD складається з наступних функціональних блоків:

– Навігаційне меню: Файл; Інструменти; Параметри; Довідка.

– Функції представлені у графічному вигляді (іконки).

– Вікна обрання групи.

– Вікно виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

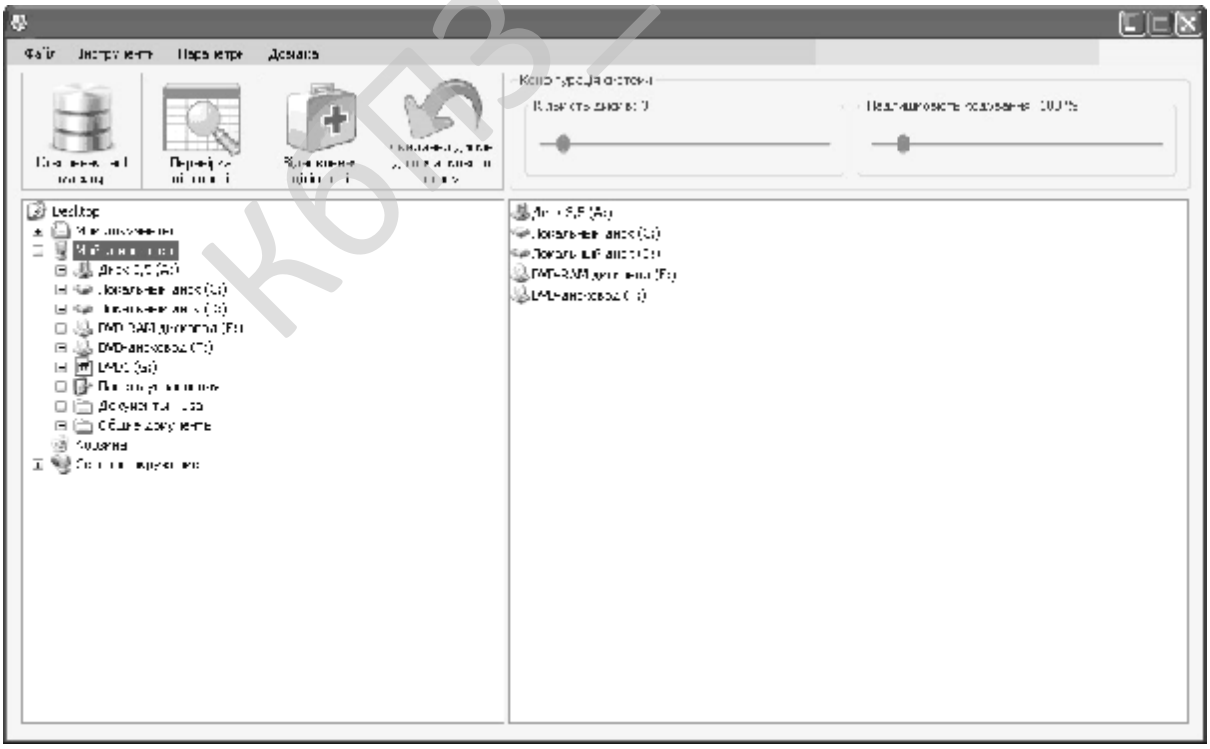


Рисунок 5.1 – Головне вікно розробленого ПЗ

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

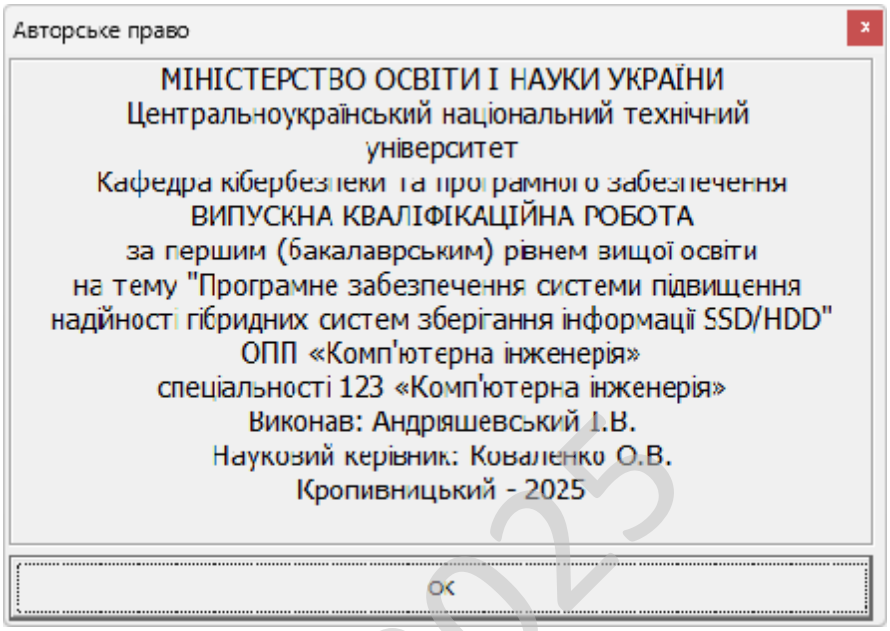


Рисунок 5.2 – Вікно розробника ПЗ

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи підвищення надійності гібридних систем зберігання інформації SSD/HDD.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем підвищення надійності гібридних систем зберігання інформації SSD/HDD.

– Досліджена система підвищення надійності гібридних систем зберігання інформації SSD/HDD.

– На основі отриманих результатів досліджень створена програмна реалізація системи підвищення надійності гібридних систем зберігання інформації SSD/HDD.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання підвищення надійності гібридних систем зберігання інформації SSD/HDD.

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 підвищення надійності гібридних систем зберігання інформації SSD/HDD. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм TEA.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Maurício Aniche. Effective Software Testing. Manning Publications. 2021. 372 p

2. Priscila Heller. Automating Workflows with GitHub Actions. Packt Publishing. 2021. 216 p.

3. JJ Geewax. API Design Patterns. Manning Publications Co. 2021. 481 p.

4. Prateek Prasad. App Design Apprentice. Razeware LLC. 2020. 272 p.

5. Dawn Griffiths, David Griffiths. Head First Android Development. O'Reilly Media, Inc. 2021. 1414 p.

6. Nathan Metzler. Kotlin Programming for Beginners. Independently published. 2021. 158 p.

7. Aaron Torres. Go Programming Cookbook Second Edition. Packt Publishing Ltd. 2019. 427 p.

8. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В.Ф., 2019. – 156 с.

9. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

10. Knuth D. Art of Computer Programming, Vol. 2: Seminumerical Algorithms 3rd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 672 p.

11. Knuth D. The Art of Computer Programming: Vol. 3: Sorting and Searching 2nd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 800 p.

12. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.

13. Вінтенко Б.Ю., Смірнов О.А., Миронець І.В., Смірнова Т.В. «Методи забезпечення відмовостійкості інтелектуальних систем підтримки

оператора». *VIII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології",* м. Кропивницький. 24-25 квітня 2025 р. – Кропивницький: ЦНТУ. – 2025. – С. 44-46.

14. Смірнов, О.А., Константинова, Л.В., Конопліцька-Слободенюк, О.К., Козірова, Н.В, Якименко, Н.М., Доренський, О.П., Буравченко, К.О. «Дослідження інструментів штучного інтелекту для роботи з базами даних та аналізу даних». *Кібербезпека: освіта, наука, техніка.* 2025. №3(27), С. 429–448.)

15. Smirnov O., Fedorov E., Neskorodieva A., Neskorodieva T. «Intellectual Classification method of Gymnastic Elements Based on Combinations of Descriptive and Generative Approache». *CEUR Workshop Proceedings* Volume 3664, 2024, Pages 11-23.

16. Вінтенко, Б., Миронець, І., Смірнов, О., Коваленко, А., Конопліцька-Слободенюк, О., Смірнова, Т., Константинова, Л. «Дослідження застосування систем підтримки оперативного персоналу об'єкту критичної інфраструктури при керуванні енергоблоком АЕС з реактором типу ВВЕР-1000». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»,* 2024. № 2(26), С. 6-26.

17. 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.

18. Kuznetsov O., Ilchenko O., Kryvinska N., Buravchenko K., Smirnov O., Savchenko Iu. «An Empirical Assessment of Leading Blockchain Financial Services». *2023 IEEE 1st Ukrainian Distributed Ledger Technology Forum (UADLTF),* Kyiv, Ukraine, 2023, pp. 1-6,

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

20. Malyukov V., Bebeshko B., Lakhno V., Smirnov O., Malyukova I., Mohylnyi H. «Managing the Purchase-Sale Process of Digital Currencies Under Fuzzy Conditions». *Lecture Notes in Networks and Systems,* 2023, 729 LNNS, pp. 104–112.

21. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user e10/11erience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

22. 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.

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

24. Вінтенко Б.Ю., Смірнов О.А., Коваленко А.С., Смірнов С.А., Буравченко К.О. «Дослідження вимог міжнародних стандартів IEC60880 та IEC62138 з розробки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 3(73), С. 155-166.

25. 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.

26. 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.

27. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.

28. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А. «Дослідження нормативної документації та стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». VI міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології", м. Кропивницький. 20-21 квітня 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 35-36.

29. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.

30. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.

31. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143

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

33. 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.

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

35. 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.

36. 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.

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

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

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

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

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

42. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P. 707-712.

43. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P.701-706.

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

45. Смірнов, О.А., Усік П.С., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю. «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». Проблеми телекомунікацій. № 1(26). С. 83-96. 2020.

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

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

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

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

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

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

52. O. Smirnov, O. Kovalenko, A. Kovalenko, S. Smirnov, V. Vialkova. The mathematical model of the testing technology for DOM XSS vulnerabilities. Scientific & practical cyber security journal (SPCSJ) Vol 2 Issue 1, 22-28 pp. [Электронный Журнал]. Georgia. Tbilisi: SCSA – 2018.

53. Oleksii Smirnov, Oleksandr Kovalenko, Jamil Al-Azzeh, Anna Kovalenko, Serhii Smirnov. Qualitative risk analysis of software development. Asian Journal of Information Technology. – Volume 17(3). – Medwell Journals. – 2018. – P. 218-230.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи підвищення надійності гібридних систем зберігання інформації SSD/HDD.

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. № 47-02 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 підвищення надійності гібридних систем зберігання інформації SSD/HDD.

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 підвищення надійності гібридних систем зберігання інформації SSD/HDD;

– 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 – 81 аркуш.

8 Етапи розробки

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

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

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

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

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 5.06.2025 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

_____ Коваленко О.В.

Програмне забезпечення системи підвищення надійності гібридних систем зберігання інформації SSD/HDD

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 72

Літера: РП

Кропивницький – 2025 року

Файл FileAnalyzer.cs - контроль цілісності даних гібридних систем зберігання інформації SSD/HDD

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

namespace RecoveryDisk
{
    /// <summary>
    /// Клас контролю цілісності даних гібридних систем зберігання інформації
SSD/HDD
    /// </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 "Множина 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 (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 (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 (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 гібридних
систем зберігання інформації SSD/HDD
        /// </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 (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 (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
    }
}
```

Файл RSRaidSSDBase.cs – робота з RAID масивами

```
using System;
using System.Threading;

namespace RecoveryStar
{
    /// <summary>
    /// Клас базової частини RAID
    /// </summary>
    public abstract class RSRaidSSDBase
    {
        #region Delegates

        /// <summary>
        /// Делегат відновлення процесу формування матриці "FLog"
        /// </summary>
        public OnUpdateDoubleValueHandler OnUpdateRSMatrixFormingProgress;

        /// <summary>
        /// Делегат завершення процесу формування матриці "FLog"
        /// </summary>
        public OnEventHandler OnRSMatrixFormingFinish;

        #endregion Delegates

        #region Public Properties & Data

        /// <summary>
        /// Екземпляр класу зайнятий обробкою?
        /// </summary>
        public bool InProcessing
        {
            get
            {
                if (
                        (this.thrRSMatrixForming != null)
                        &&
                        (
                            (this.thrRSMatrixForming.ThreadState ==
ThreadState.Running)
                        ||
                            (this.thrRSMatrixForming.ThreadState ==
ThreadState.WaitSleepJoin)
                        )
                    )
                {
                    return true;

                } else
                {
                    return false;
                }
            }
        }

        /// <summary>
        /// Екземпляр класу сконфігурований коректно?
        /// </summary>
        public bool ConfigIsOK
        {
            get
            {
                if (!InProcessing)
                {
                    return this.configIsOK;
```

```
        } else
        {
            return false;
        }
    }
}

/// <summary>
/// Екземпляр класу ініціалізований коректно (придатний до роботи)?
/// </summary>
protected bool configIsOK;

/// <summary>
/// Булева 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>
protected bool finished;

/// <summary>
/// Кількість основних томів
/// </summary>
public int DataCount
{
    get
    {
        if (!InProcessing)
        {
            return this.n;

        } else
        {
            return -1;
        }
    }
}

/// <summary>
/// Кількість основних томів
/// </summary>
protected int n;

/// <summary>
/// Кількість томів для відновлення
/// </summary>
public int EccCount
{
    get
    {
        if (!InProcessing)
```

```
                {
                    return this.m;

                } else
                {
                    return -1;
                }
            }
        }

        /// <summary>
        /// Кількість томів для відновлення
        /// </summary>
        protected int m;

        /// <summary>
        /// Тип кодека (за типом використовуваної матриці)
        /// </summary>
        public int CodecType
        {
            get
            {
                if (!InProcessing)
                {
                    return this.eRSType;

                } else
                {
                    return -1;
                }
            }
        }

        /// <summary>
        /// Тип кодека Ріда-Соломона (за типом використовуваної матриці
кодування)
        /// </summary>
        protected int eRSType;

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

            set
            {
                if (
                        (this.thrRSMatrixForming != null)
                     &&
                        (this.thrRSMatrixForming.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.thrRSMatrixForming.Priority = this.threadPriority;
                }
            }
        }

        /// <summary>
        /// Пріоритет процесу підготовки матриці кодування
        /// </summary>
        protected ThreadPriority threadPriority;

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

        /// <summary>
        /// Подія, що виконується по завершенню обробки
        /// </summary>
        protected ManualResetEvent[] finishedEvent;

        #endregion Public Properties & Data

        #region Data

        // Об'єкт класу роботи з елементами поля Галуа
        protected GF16 eGF16;

        // Матриця RAID-подібного кодера Ріда-Соломона
        protected int[] FLog;
```

```
        // Дисперсна матриця (використовується замість "A")
        protected int[] D;

        // "Альтернативна" матриця (використовується замість "D")
        protected int[] A;

        // Основна конфігурація змінилася?
        protected bool mainConfigChanged;

        // Кількість ітерацій першої й другої стадій підготовки матриці
кодування
        protected double iterOfFirstStage;
        protected double iterOfSecondStage;

        // Потік заповнення матриці "FLog" перед виконанням кодування /
декодування
        protected Thread thrRSMatrixForming;

        // Подія припинення підготовки матриці кодування
        protected ManualResetEvent[] exitEvent;

        // Подія продовження підготовки матриці кодування
        protected ManualResetEvent[] executeEvent;

        #endregion Data

        #region Construction & Destruction

        /// <summary>
        /// Конструктор базового класу сутності "RAID-подібний кодек Ріда-
Соломона"
        /// </summary>
        public RSRaidSSDBase()
        {
            // Створюємо екземпляр класу для роботи з арифметикою поля Галуа
(2^16)
            this.eGF16 = new GF16();

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

            // Основна конфігурація змінилася?
            this.mainConfigChanged = true;

            // Екземпляр класу ініціалізовано коректно (придатний до роботи)?
            this.configIsOK = false;

            // По-замовчанню встановлюється фоновий пріоритет
            this.threadPriority = 0;

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Запуск процесу заповнення матриці "FLog" даними
```

```
        /// </summary>
        /// <param name="runAsSeparateThread">Запускати в окремому
потоці?</param>
        /// <returns>Булевий прапор операції</returns>
        public bool Prepare(bool runAsSeparateThread)
        {
            // Якщо потік формування матриці "FLog" працює - не дозволяємо
повторний запуск
            if (InProcessing)
            {
                return false;
            }

            // Якщо конфігурація встановлена некоректно - виходимо
            if (!this.configIsOK)
            {
                return false;
            }

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

            // Скидаємо подію завершення обробки
            this.finishedEvent[0].Reset();

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

            // Якщо зазначено, що не потрібен запуск в окремому потоці,
            // запускаємо в даному
            if (!runAsSeparateThread)
            {
                // Заповнюємо матрицю кодування
                FillFLog();

                // Повертаємо результат обробки
                return this.configIsOK;
            }

            // Створюємо потік формування матриці "FLog"...
            this.thrRSMatrixForming = new Thread(new ThreadStart(FillFLog));

            //...потім даємо йому ім'я...
            this.thrRSMatrixForming.Name = "RSRaidSSD.FillFLog()";

            //...встановлюємо обраний пріоритет завдання...
            this.thrRSMatrixForming.Priority = this.threadPriority;

            //...і запускаємо
            this.thrRSMatrixForming.Start();

            return true;
        }

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

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

        /// <summary>
        /// Постановка потоку обробки на паузу
```

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

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

        #endregion Public Operations

        #region Protected Operations

        /// <summary>
        /// Нормалізація значень "n" і "m" з метою запобігання переповнення
змінних,
        /// ітерацій, що зберігають загальну кількість
        /// </summary>
        protected void NormalizeNM(ref double n, ref double m)
        {
            double maxVal = 0;

            if (n > m)
            {
                maxVal = n;

            } else
            {
                maxVal = m;
            }

            double divider = maxVal / 100.0;

            if (divider > 1)
            {
                n /= divider;
                m /= divider;
            }
        }

        /// <summary>
        /// Метод пошуку індексу рядка,
        /// </summary>
        /// <param name="rowNum">Номер рядка</param>
        /// <returns>Індекс рядка, придатної для заміни</returns>
        protected int FindSwapRow(int rowNum)
        {
            // Пробігаємо по всіх наявних рядках матриці
            // у зазначеному стовпці
            for (int i = rowNum; i < (this.n + this.m); i++)
            {
                if (this.D[(i * this.n) + rowNum] != 0)
                {
                    return i;
                }
            }

            return -1;
        }

        /// <summary>
        /// Метод перестановки двох рядків місцями
```

```
        /// </summary>
        /// <param name="rowNum1">Індекс першого рядка</param>
        /// <param name="rowNum2">Індекс другого рядка</param>
        protected void SwapRows(int rowNum1, int rowNum2)
        {
            // Обчислюємо зсув до елементів i-ой рядка
            int rowNum1this_n = rowNum1 * this.n;
            int rowNum2this_n = rowNum2 * this.n;

            for (int j = 0; j < this.n; j++)
            {
                int dIdx1 = rowNum1this_n + j;
                int dIdx2 = rowNum2this_n + j;

                int tmp       = this.D[dIdx1];
                this.D[dIdx1] = this.D[dIdx2];
                this.D[dIdx2] = tmp;
            }
        }

        /// <summary>
        /// Метод одержання дисперсної матриці "D"
        /// </summary>
        /// <returns>Булевий прапор результату операції</returns>
        protected bool MakeDispersalMatrix()
        {
            // Виділяємо пам'ять під матрицю "FLog"
            this.D = new int[(this.n + this.m) * this.n];

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

                // Обчислення рядка матриці Вандермонда (цей блок обчислень
                // може бути реалізований і без використання функції зведення
                // елемента в ступінь, але поточна реалізація припускає більшу
                // гнучкість і зрозумілість)
                for (int j = 0; j < this.n; j++)
                {
                    this.D[i_n + j] = this.eGF16.Pow(i, j);
                }
            }

            // Обчислюємо розподіл відсотків ітерацій по стадіях для
            // коректної обробки відсотків
            double allStageIter  = this.iterOfFirstStage +
this.iterOfSecondStage;
            int percOfFirstStage = (int)((100.0 * this.iterOfFirstStage) /
allStageIter);

            // Дана стадія повинна займати хоча б один відсоток
            // (для коректності розрахунків)
            if (percOfFirstStage == 0)
            {
                percOfFirstStage = 1;
            }

            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = this.n / percOfFirstStage;

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

```
                progressMod1 = 1;
            }

            // Цикл вибору діагонального елемента
            for (int k = 1; k < this.n; ++k)
            {
                // Шукаємо рядок, у якій елемент на головній
                // діагоналі міг би бути ненульовим
                int swapIdx = FindSwapRow(k);

                // Якщо підходящий рядок не може бути знайдений -
                // це помилка - ...
                if (swapIdx == -1)
                {
                    //...вказуємо на помилку конфігурації
                    this.configIsOK = false;

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

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

                    return false;
                }

                // Якщо був знайдений рядок, відмінна від поточної...
                if (swapIdx != k)
                {
                    //...міняємо рядки місцями
                    SwapRows(swapIdx, k);
                }

                int k_n = k * this.n;

                // Витягаємо діагональний елемент
                int diagElem = this.D[k_n + k];

                // Якщо діагональний елемент не дорівнює "1", множимо весь
стовпець
                // на зворотний йому елемент, перетворюючи діагональний в "1"
                if (diagElem != 1)
                {
                    // Обчислюємо зворотний елемент для "diagElem"
                    int diagElemInv = this.eGF16.Inv(diagElem);

                    // Робимо необхідну обробку елементів стовпця -
                    // множимо його на елемент, зворотний "diagElem"
                    for (int i = k; i < (this.n + this.m); i++)
                    {
                        int dIdx = (i * this.n) + k;

                        this.D[dIdx] = this.eGF16.Mul(this.D[dIdx],
diagElemInv);
                    }
                }

                // Для всіх стовпців...
                for (int j = 0; j < this.n; j++)
                {
                    // Витягаємо множник поточного стовпця
                    int colMult = this.D[k_n + j];

                    //...не є стовпцями розв'язного елемента...
                    if (
                            (j != k)
                         &&
                            (colMult != 0)
                        )
```

```
                    {
                        for (int i = k; i < (this.n + this.m); i++)
                        {
                            int i_n  = i * this.n;
                            int dIdx = i_n + j;

                            //...робимо заміну Cj = Cj - Dk,j * Ck
                            this.D[dIdx] = this.D[dIdx] ^
this.eGF16.Mul(colMult, this.D[i_n + k]);
                        }
                    }
                }

                // Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((k % progressMod1) == 0)
                     &&
                        (OnUpdateRSMatrixFormingProgress != null)
                   )
                {
                    //...виводимо дані
                    OnUpdateRSMatrixFormingProgress(((double)(k + 1) /
(double)this.n) * 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))
                {
                    // Вказуємо, що декодер не сконфігурований коректно
                    this.configIsOK = false;

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

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

                    return false;
                }
            }

            return true;
        }

        /// <summary>
        /// Метод одержання "альтернативної" матриці "A" : у ньому для
заповнення матриці кодування
        /// з 65535 констант вибираються 32768, таких, щоб логарифм кожної з них
був взаємно
        /// простим зі значенням "65535", тобто щоб їх НСД (найбільший спільний
дільник) був рівний
        /// "1". Порушення цієї умови приводить до неможливості обігу матриці
кодування,
        /// і, відповідно, до неможливості відновлення даних гібридних систем
зберігання інформації SSD/HDD)
        /// </summary>
        /// <returns>Булевий прапор результату операції</returns>
        protected bool MakeAlternativeMatrix()
        {
            // Перебирається значення логарифму з метою подальшого одержання
константи
            // для занесення в матрицю шляхом його потенцирування
            int logBase = 0;
```

```
            // Відновлення по "logBase" підстановка ступеня для формування рядка
            // матриці Вандермонда
            int powBase = 0;

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

            // Обчислюємо розподіл відсотків ітерацій по стадіях для
            // коректної обробки відсотків
            double allStageIter  = this.iterOfFirstStage +
this.iterOfSecondStage;
            int percOfFirstStage = (int)((100.0 * this.iterOfFirstStage) /
allStageIter);

            // Дана стадія повинна займати хоча б один відсоток
            // (для коректності розрахунків)
            if (percOfFirstStage == 0)
            {
                percOfFirstStage = 1;
            }

            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = this.m / percOfFirstStage;

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

            // Заповнюємо матрицю даними (формуємо матрицю Вандермонда)
            for (int i = 0; i < this.m; i++)
            {
                // Поки "logBase" не взаємно просто з "65535"...
                while (
                          ((logBase % 3)   == 0)
                       ||
                          ((logBase % 5)   == 0)
                       ||
                          ((logBase % 17)  == 0)
                       ||
                          ((logBase % 257) == 0)
                      )
                {
                    ++logBase;
                }

                //...потім, відновлюємо його значення...
                powBase = this.eGF16.Exp(logBase++);

                // Зсув у масиві до елементів i-ой рядка
                int i_n = i * this.n;

                for (int j = 0; j < this.n; j++)
                {
                    //...i використовуємо для формування рядка матриці
Вандермонда
                    this.A[i_n + j] = this.eGF16.Pow(powBase, j);
                }

                // Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((i % progressMod1) == 0)
                     &&
                        (OnUpdateRSMatrixFormingProgress != null)
```

```
                    )
                {
                    //...виводимо дані
                    OnUpdateRSMatrixFormingProgress(((double)(i + 1) /
(double)this.m) * 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))
                {
                    // Вказуємо, що декодер не сконфігуровано коректно
                    this.configIsOK = false;

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

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

                    return false;
                }
            }

            return true;
        }

        /// <summary>
        /// Заповнення матриці "FLog" даними
        /// </summary>
        protected virtual void FillFLog() { }

        #endregion Protected Operations
    }
}
```

Файл RSRaidSSDEncoder.cs - кодування алгоритмом Ріда-Соломона

```
using System;
using System.Threading;

namespace RecoveryDisk
{
    /// <summary>
    /// Клас RAID-подібного кодера Ріда-Соломона
    /// </summary>
    public class RSRaidSSDEncoder : RSRaidSSDBase
    {
        #region Construction & Destruction

        /// <summary>
        /// Конструктор кодера за замовчуванням
        /// </summary>
        public RSRaidSSDEncoder()
        {
            // 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 RSRaidSSDEncoder(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 (по типу
матриці)</param>
        public RSRaidSSDEncoder(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 (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
    }
}
```

Файл RSRaidSSDDecoder.cs - декодування алгоритмомом Ріда-Соломона

```
using System;
using System.Threading;

namespace RecoveryDisk
{
    /// <summary>
    /// Клас RAID-подібного декодера Ріда-Соломона
    /// </summary>
    public class RSRaidSSDDecoder : RSRaidSSDBase
    {
        #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 RSRaidSSDDecoder()
        {
            // 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 RSRaidSSDDecoder(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 (по типу
матриці)</param>
        public RSRaidSSDDecoder(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 (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 i-ой рядка
                    int i_n = (i * this.n);

                    for (int j = 0; j < this.n; j++)
```

```
                    {
                        int idx = i_n + j;

                        this.FLog[idx] = this.eGF16.Mul(this.FLog[idx],
pivotInv);
                    }
                }

                // Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((k % progressMod1) == 0)
                     &&
                        (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 MainForm.cs - головне вікно програми

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

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

        #region Windows Form Designer generated code

        /// <summary>
        /// Викликаємо метод для підтримки інтерфейсу
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.Windows.Forms.TreeNode treeNode1 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode2 = new
System.Windows.Forms.TreeNode("Documents and Settings", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode1});
            System.Windows.Forms.TreeNode treeNode3 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode4 = new
System.Windows.Forms.TreeNode("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);
            //
            // helpToolStripMenuItem
            //
            this.helpToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.separatorToolStripMenuItem,
            this.aboutToolStripMenuItem});
            this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
            this.helpToolStripMenuItem.Size = new System.Drawing.Size(60, 17);
            this.helpToolStripMenuItem.Text = "Довідка";
            //
            // separatorToolStripMenuItem
            //
            this.separatorToolStripMenuItem.Name = "separatorToolStripMenuItem";
            this.separatorToolStripMenuItem.Size = new System.Drawing.Size(163,
6);
            //
```

```
            // aboutToolStripMenuItem
            //
            this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
            this.aboutToolStripMenuItem.Size = new System.Drawing.Size(166, 22);
            this.aboutToolStripMenuItem.Text = "Про програму...";
            this.aboutToolStripMenuItem.Click += new
System.EventHandler(this.aboutToolStripMenuItem_Click);
            //
            // coderConfigGroupBox
            //
            this.coderConfigGroupBox.Controls.Add(this.redundancyGroupBox);
            this.coderConfigGroupBox.Controls.Add(this.allVolCountGroupBox);
            this.coderConfigGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.coderConfigGroupBox.Location = new System.Drawing.Point(414,
26);
            this.coderConfigGroupBox.Name = "coderConfigGroupBox";
            this.coderConfigGroupBox.Size = new System.Drawing.Size(561, 98);
            this.coderConfigGroupBox.TabIndex = 5;
            this.coderConfigGroupBox.TabStop = false;
            this.coderConfigGroupBox.Text = "Конфігурація системи";
            //
            // redundancyGroupBox
            //
            this.redundancyGroupBox.Controls.Add(this.redundancyMacTrackBar);
            this.redundancyGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.redundancyGroupBox.Location = new System.Drawing.Point(286,
21);
            this.redundancyGroupBox.Name = "redundancyGroupBox";
            this.redundancyGroupBox.Size = new System.Drawing.Size(264, 65);
            this.redundancyGroupBox.TabIndex = 4;
            this.redundancyGroupBox.TabStop = false;
            this.redundancyGroupBox.Text = "Надлишковість кодування";
            //
            // allVolCountGroupBox
            //
            this.allVolCountGroupBox.Controls.Add(this.allVolCountMacTrackBar);
            this.allVolCountGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.allVolCountGroupBox.Location = new System.Drawing.Point(12,
21);
            this.allVolCountGroupBox.Name = "allVolCountGroupBox";
            this.allVolCountGroupBox.Size = new System.Drawing.Size(264, 65);
            this.allVolCountGroupBox.TabIndex = 3;
            this.allVolCountGroupBox.TabStop = false;
            this.allVolCountGroupBox.Text = "Кількість дисків";
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // redundancyMacTrackBar
            //
            this.redundancyMacTrackBar.BackColor =
System.Drawing.Color.Transparent;
            this.redundancyMacTrackBar.BorderColor =
System.Drawing.SystemColors.ActiveBorder;
            this.redundancyMacTrackBar.Font = new System.Drawing.Font("Verdana",
8.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.redundancyMacTrackBar.ForeColor =
System.Drawing.Color.FromArgb(((int)(((byte)(123)))), ((int)(((byte)(125)))),
((int)(((byte)(123)))));
            this.redundancyMacTrackBar.IndentHeight = 6;
```

```
            this.redundancyMacTrackBar.Location = new System.Drawing.Point(6,
24);
            this.redundancyMacTrackBar.Maximum = 199;
            this.redundancyMacTrackBar.Minimum = 0;
            this.redundancyMacTrackBar.Name = "redundancyMacTrackBar";
            this.redundancyMacTrackBar.Size = new System.Drawing.Size(252, 28);
            this.redundancyMacTrackBar.TabIndex = 6;
            this.redundancyMacTrackBar.TextTickStyle =
System.Windows.Forms.TickStyle.None;
            this.redundancyMacTrackBar.TickColor =
System.Drawing.Color.FromArgb(((int)(((byte)(148)))), ((int)(((byte)(146)))),
((int)(((byte)(148)))));
            this.redundancyMacTrackBar.TickHeight = 4;
            this.redundancyMacTrackBar.TickStyle =
System.Windows.Forms.TickStyle.None;
            this.redundancyMacTrackBar.TrackerColor =
System.Drawing.Color.FromArgb(((int)(((byte)(24)))), ((int)(((byte)(130)))),
((int)(((byte)(198)))));
            this.redundancyMacTrackBar.TrackerSize = new System.Drawing.Size(10,
16);
            this.redundancyMacTrackBar.TrackLineColor =
System.Drawing.Color.FromArgb(((int)(((byte)(90)))), ((int)(((byte)(93)))),
((int)(((byte)(90)))));
            this.redundancyMacTrackBar.TrackLineHeight = 3;
            this.redundancyMacTrackBar.Value = 19;
            this.redundancyMacTrackBar.ValueChanged += new
EConTech.Windows.MACUI.ValueChangedHandler(this.redundancyMacTrackBar_ValueChang
ed);
            this.redundancyMacTrackBar.MouseUp += new
System.Windows.Forms.MouseEventHandler(this.redundancyMacTrackBar_MouseUp);
            //
            // allVolCountMacTrackBar
            //
            this.allVolCountMacTrackBar.BackColor =
System.Drawing.Color.Transparent;
            this.allVolCountMacTrackBar.BorderColor =
System.Drawing.SystemColors.ActiveBorder;
            this.allVolCountMacTrackBar.Font = new
System.Drawing.Font("Verdana", 8.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.allVolCountMacTrackBar.ForeColor =
System.Drawing.Color.FromArgb(((int)(((byte)(123)))), ((int)(((byte)(125)))),
((int)(((byte)(123)))));
            this.allVolCountMacTrackBar.IndentHeight = 6;
            this.allVolCountMacTrackBar.Location = new System.Drawing.Point(6,
24);
            this.allVolCountMacTrackBar.Maximum = 15;
            this.allVolCountMacTrackBar.Minimum = 0;
            this.allVolCountMacTrackBar.Name = "allVolCountMacTrackBar";
            this.allVolCountMacTrackBar.Size = new System.Drawing.Size(252, 28);
            this.allVolCountMacTrackBar.TabIndex = 5;
            this.allVolCountMacTrackBar.TextTickStyle =
System.Windows.Forms.TickStyle.None;
            this.allVolCountMacTrackBar.TickColor =
System.Drawing.Color.FromArgb(((int)(((byte)(148)))), ((int)(((byte)(146)))),
((int)(((byte)(148)))));
            this.allVolCountMacTrackBar.TickHeight = 4;
            this.allVolCountMacTrackBar.TickStyle =
System.Windows.Forms.TickStyle.None;
            this.allVolCountMacTrackBar.TrackerColor =
System.Drawing.Color.FromArgb(((int)(((byte)(24)))), ((int)(((byte)(130)))),
((int)(((byte)(198)))));
            this.allVolCountMacTrackBar.TrackerSize = new
System.Drawing.Size(10, 16);
            this.allVolCountMacTrackBar.TrackLineColor =
System.Drawing.Color.FromArgb(((int)(((byte)(90)))), ((int)(((byte)(93)))),
((int)(((byte)(90)))));
            this.allVolCountMacTrackBar.TrackLineHeight = 3;
            this.allVolCountMacTrackBar.Value = 2;
```

```
            this.allVolCountMacTrackBar.ValueChanged += new
EConTech.Windows.MACUI.ValueChangedHandler(this.allVolCountMacTrackBar_ValueChan
ged);
            this.allVolCountMacTrackBar.MouseUp += new
System.Windows.Forms.MouseEventHandler(this.allVolCountMacTrackBar_MouseUp);
            //
            // browser
            //
            this.browser.AutoValidate =
System.Windows.Forms.AutoValidate.EnablePreventFocusChange;
            this.browser.ListViewMode = System.Windows.Forms.View.List;
            this.browser.Location = new System.Drawing.Point(12, 131);
            this.browser.Name = "browser";
            treeNode1.Name = "";
            treeNode1.Text = "";
            treeNode2.ImageIndex = 18;
            treeNode2.Name = "backreg";
            treeNode2.SelectedImageIndex = 20;
            treeNode2.Text = "backreg";
            treeNode3.Name = "";
            treeNode3.Text = "";
            treeNode4.ImageIndex = 18;
            treeNode4.Name = "CISCO_CCNA";
            treeNode4.SelectedImageIndex = 20;
            treeNode4.Text = "CISCO_CCNA";
            treeNode5.Name = "";
            treeNode5.Text = "";
            treeNode6.ImageIndex = 18;
            treeNode6.Name = "Documents and Settings";
            treeNode6.SelectedImageIndex = 20;
            treeNode6.Text = "Documents and Settings";
            treeNode7.Name = "";
            treeNode7.Text = "";
            treeNode8.ImageIndex = 18;
            treeNode8.Name = "Downloads";
            treeNode8.SelectedImageIndex = 20;
            treeNode8.Text = "Downloads";
            treeNode9.Name = "";
            treeNode9.Text = "";
            treeNode10.ImageIndex = 18;
            treeNode10.Name = "Inprise";
            treeNode10.SelectedImageIndex = 20;
            treeNode10.Text = "Inprise";
            treeNode11.Name = "";
            treeNode11.Text = "";
            treeNode12.ImageIndex = 18;
            treeNode12.Name = "Program Files";
            treeNode12.SelectedImageIndex = 20;
            treeNode12.Text = "Program Files";
            treeNode13.ImageIndex = 33;
            treeNode13.Name = "Recycled";
            treeNode13.SelectedImageIndex = 34;
            treeNode13.Text = "Recycled";
            treeNode14.ImageIndex = 18;
            treeNode14.Name = "RECYCLER";
            treeNode14.SelectedImageIndex = 20;
            treeNode14.Text = "RECYCLER";
            treeNode15.ImageIndex = 18;
            treeNode15.Name = "System Volume Information";
            treeNode15.SelectedImageIndex = 20;
            treeNode15.Text = "System Volume Information";
            treeNode16.ImageIndex = 18;
            treeNode16.Name = "temp";
            treeNode16.SelectedImageIndex = 20;
            treeNode16.Text = "temp";
            treeNode17.Name = "";
            treeNode17.Text = "";
            treeNode18.ImageIndex = 18;
            treeNode18.Name = "WINDOWS";
```

```
            treeNode18.SelectedImageIndex = 20;
            treeNode18.Text = "WINDOWS";
            treeNode19.ImageIndex = 23;
            treeNode19.Name = "Локальный диск (C:)";
            treeNode19.SelectedImageIndex = 24;
            treeNode19.Text = "Локальный диск (C:)";
            this.browser.SelectedNode = treeNode19;
            this.browser.ShowFoldersButton = false;
            this.browser.ShowNavigationBar = false;
            this.browser.Size = new System.Drawing.Size(962, 432);
            this.browser.SplitterDistance = 398;
            this.browser.StartUpDirectoryOther = "C:\\";
            this.browser.TabIndex = 0;
            this.browser.Load += new System.EventHandler(this.browser_Load);
            //
            // testButton
            //
            this.testButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.testButton.Image =
global::RecoveryStar.Properties.Resources.table_sql_view32451;
            this.testButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.testButton.Location = new System.Drawing.Point(111, 27);
            this.testButton.Name = "testButton";
            this.testButton.Size = new System.Drawing.Size(100, 97);
            this.testButton.TabIndex = 4;
            this.testButton.Text = "Перевірка цілісності";
            this.testButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.testButton.UseVisualStyleBackColor = true;
            this.testButton.Click += new
System.EventHandler(this.testButton_Click);
            //
            // repairButton
            //
            this.repairButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.repairButton.Image =
global::RecoveryStar.Properties.Resources.medical_bag465471;
            this.repairButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.repairButton.Location = new System.Drawing.Point(210, 27);
            this.repairButton.Name = "repairButton";
            this.repairButton.Size = new System.Drawing.Size(100, 97);
            this.repairButton.TabIndex = 3;
            this.repairButton.Text = "Відновлення цілісності";
            this.repairButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.repairButton.UseVisualStyleBackColor = true;
            this.repairButton.Click += new
System.EventHandler(this.repairButton_Click);
            //
            // recoverButton
            //
            this.recoverButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.recoverButton.Image =
global::RecoveryStar.Properties.Resources.redo6786987;
            this.recoverButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.recoverButton.Location = new System.Drawing.Point(309, 27);
            this.recoverButton.Name = "recoverButton";
            this.recoverButton.Size = new System.Drawing.Size(100, 97);
            this.recoverButton.TabIndex = 2;
            this.recoverButton.Text = "Скидання дисків до початкового стану";
            this.recoverButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.recoverButton.UseVisualStyleBackColor = true;
            this.recoverButton.Click += new
System.EventHandler(this.recoverButton_Click);
            //
```

```
            // protectButton
            //
            this.protectButton.BackColor = System.Drawing.SystemColors.Control;
            this.protectButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.protectButton.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8.25F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.protectButton.Image =
global::RecoveryStar.Properties.Resources.Database_1_64x64e65768;
            this.protectButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.protectButton.Location = new System.Drawing.Point(12, 27);
            this.protectButton.Name = "protectButton";
            this.protectButton.Size = new System.Drawing.Size(100, 97);
            this.protectButton.TabIndex = 1;
            this.protectButton.Text = "Створення RaidSSD масиву";
            this.protectButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.protectButton.UseVisualStyleBackColor = false;
            this.protectButton.Click += new
System.EventHandler(this.protectButton_Click);
            //
            // ToolStripMenuItem
            //
            this.ToolStripMenuItem.Image =
global::RecoveryStar.Properties.Resources.Exit;
            this.ToolStripMenuItem.Name = "ToolStripMenuItem";
            this.ToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
            this.ToolStripMenuItem.Text = "Вихід";
            this.ToolStripMenuItem.Click += new
System.EventHandler(this.выходToolStripMenuItem_Click);
            //
            // ToolStripMenuItem
            //
            this.тестБыстродействияToolStripMenuItem.Image =
global::RecoveryStar.Properties.Resources.StartBenchmark;
            this.тестБыстродействияToolStripMenuItem.ImageScaling =
System.Windows.Forms.ToolStripItemImageScaling.None;
            this.ToolStripMenuItem.Name = "ToolStripMenuItem";
            this.ToolStripMenuItem.Size = new System.Drawing.Size(161, 22);
            this.ToolStripMenuItem.Text = "Тест швидкодії";
            this.ToolStripMenuItem.Click += new
System.EventHandler(this.тестБыстродействияToolStripMenuItem_Click);
            //
            // MainForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(986, 576);
            this.Controls.Add(this.coderConfigGroupBox);
            this.Controls.Add(this.testButton);
            this.Controls.Add(this.repairButton);
            this.Controls.Add(this.recoverButton);
            this.Controls.Add(this.protectButton);
            this.Controls.Add(this.browser);
            this.Controls.Add(this.menuStrip);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.MainMenuStrip = this.menuStrip;
            this.MaximizeBox = false;
            this.Name = "MainForm";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Гарантоване збереження інформації на основі RAID
масивів";
            this.Load += new System.EventHandler(this.MainForm_Load);
```

```
            this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.MainForm_FormClosing);
            this.menuStrip.ResumeLayout(false);
            this.menuStrip.PerformLayout();
            this.coderConfigGroupBox.ResumeLayout(false);
            this.redundancyGroupBox.ResumeLayout(false);
            this.redundancyGroupBox.PerformLayout();
            this.allVolCountGroupBox.ResumeLayout(false);
            this.allVolCountGroupBox.PerformLayout();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.MenuStrip menuStrip;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.Button protectButton;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.ToolStripSeparator
separatorToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.Button repairButton;
        private System.Windows.Forms.Button testButton;
        private System.Windows.Forms.GroupBox coderConfigGroupBox;
        private System.Windows.Forms.GroupBox redundancyGroupBox;
        private System.Windows.Forms.GroupBox allVolCountGroupBox;
        private EConTech.Windows.MACUI.MACTrackBar allVolCountMacTrackBar;
        private EConTech.Windows.MACUI.MACTrackBar redundancyMacTrackBar;
        private System.Windows.Forms.ToolTip toolTip;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        internal FileBrowser.Browser browser;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem ToolStripMenuItem;
        private System.Windows.Forms.Button recoverButton;
    }
}
```

Файл ProcessForm.cs - вікно створення та перевірки RaidSSD масивів

```
namespace RecoveryDisk
{
    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 гібридних систем зберігання інформації SSD/HDD";
            //
            // 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;
    }
}
```

Файл BenchmarkForm.cs - вікно тестування швидкодії алгоритму

```
      namespace RecoveryDisk
{
    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 RecoveryStar
{
    partial class AboutForm
    {
        /// <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.developersListBoxdevelopersListBox = new
System.Windows.Forms.ListBox();
            this.RSIconTimer = new System.Windows.Forms.Timer(this.components);
            this.okButtonXP = new PinkieControls.ButtonXP();
            this.SuspendLayout();
            //
            // developersListBoxdevelopersListBox
            //
            this.developersListBoxdevelopersListBox.BackColor =
System.Drawing.SystemColors.Control;
            this.developersListBoxdevelopersListBox.BorderStyle =
System.Windows.Forms.BorderStyle.None;
            this.developersListBoxdevelopersListBox.FormattingEnabled = true;
            this.developersListBoxdevelopersListBox.Items.AddRange(new object[]
{
            "БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА",
            "",
            "На тему:",
            "",
            "Програмне забезпечення підвищення надійності гібридних систем
зберігання інформації SSD/HDD",
            " ",
            "",
            "",
            "Керівник: Коваленко О.В.",
            "",
            "Розробив: студент Андріяшевський Іван Віталійович ",
            "                  гр. КІ-21-2                                    ",
            "",
            "м. Кропивницький 2025"});
            this.developersListBoxdevelopersListBox.Location = new
System.Drawing.Point(11, 6);
```

```
            this.developersListBoxdevelopersListBox.Name =
"developersListBoxdevelopersListBox";
            this.developersListBoxdevelopersListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.developersListBoxdevelopersListBox.Size = new
System.Drawing.Size(330, 182);
            this.developersListBoxdevelopersListBox.TabIndex = 0;
            this.developersListBoxdevelopersListBox.TabStop = false;
            this.developersListBoxdevelopersListBox.SelectedIndexChanged += new
System.EventHandler(this.developersListBoxdevelopersListBox_SelectedIndexChanged
);
            //
            // RSIconTimer
            //
            this.RSIconTimer.Interval = 40;
            this.RSIconTimer.Tick += new
System.EventHandler(this.RSIconTimer_Tick);
            //
            // okButtonXP
            //
            this.okButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.okButtonXP.DefaultScheme = true;
            this.okButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.okButtonXP.Hint = "";
            this.okButtonXP.Location = new System.Drawing.Point(266, 177);
            this.okButtonXP.Name = "okButtonXP";
            this.okButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.okButtonXP.Size = new System.Drawing.Size(75, 23);
            this.okButtonXP.TabIndex = 0;
            this.okButtonXP.Text = "OK";
            this.okButtonXP.Click += new
System.EventHandler(this.okButtonXP_Click);
            //
            // AboutForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(350, 212);
            this.Controls.Add(this.okButtonXP);
            this.Controls.Add(this.developersListBoxdevelopersListBox);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "AboutForm";
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "Про програму...";
            this.Load += new System.EventHandler(this.AboutForm_Load);
            this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.AboutForm_FormClosing);
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.ListBox developersListBoxdevelopersListBox;
        private System.Windows.Forms.Timer RSIconTimer;
        private PinkieControls.ButtonXP okButtonXP;

    }
}
```