Центральноукраїнський національний технічний університет
Центр заочної та дистанційної освіти
Кафедра кібербезпеки та програмного забезпечення

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи масштабованої розподіленої файлової системи”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-23Мз
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Лівітчук О.В.
«____» _____ 2024 р.

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

м. Кропивницький

Центральноукраїнський національний технічний університет

Центр *Заочної та дистанційної освіти*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

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
« 6 » вересня 2024 року

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

Лівітчуку Олексію Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи масштабованої розподіленої файлової системи*

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

затверджені наказом вищого навчального закладу № 21-13 від 07.08.2024 року

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

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи масштабованої розподіленої файлової системи*

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

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

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

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

6. Консультанти розділів роботи

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

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

КАЛЕНДАРНИЙ ПЛАН

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

Дата видачі завдання
« 6 » вересня 2024 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2024 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Лівітчук О.В. Дослідження та програмна реалізація системи масштабованої розподіленої файлової системи. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи масштабованої розподіленої файлової системи.

Метою розробки є дослідження та програмна реалізація системи масштабованої розподіленої файлової системи.

Об'єктом дослідження є процес масштабованої розподіленої файлової системи.

Предметом дослідження є методи масштабованої розподіленої файлової системи.

Методи дослідження базуються на методах файлових систем, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи масштабованої розподіленої файлової системи.

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, розподілена файлова система

ABSTRACT

Livitchuk O.V. Research and software implementation of a scalable distributed file system. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this final qualification work for the second (master's) level of higher education, software is developed, which is intended for a scalable distributed file system.

The purpose of the development is research and software implementation of a scalable distributed file system.

The object of research is the process of a scalable distributed file system.

The subject of research is the methods of a scalable distributed file system.

Research methods are based on file system methods, mathematical statistics methods, and software development methods.

The result of the work is a software implementation of a scalable distributed file system.

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

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

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

The program was developed in the Visual C# environment.

Keywords: computer engineering, distributed file system

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

БЧХ	–	Боуз – Чоудхурі – Хоквінгхем
ЕОМ	–	електронно – обчислювальна машина
ЗП	–	запам’ятовуючий пристрій
КЗПІ	–	канал зберігання й передачі інформації
МНЗІ	–	методи підвищення надійності зберігання інформації
ПЕОМ	–	персональна електронно – обчислювальна машина
ПЗ	–	програмне забезпечення
СЗД	–	система зберігання даних
СНЗІ	–	система підвищення надійності зберігання інформації
ESRM	–	керування корпоративними ресурсами зберігання
SRM	–	керування ресурсами зберігання
SAN	–	адміністрування мереж

ВСТУП

Актуальність теми. Як економічно вигідне сховище неструктурованих або напівструктурованих даних підходять і програмно обумовлені рішення.

Для зберігання даних підприємствам потрібні добре масштабовані модернізуємі рішення, але зовсім не обов'язково здобувати дорогі системи SAN. Як економічно вигідне сховище неструктурованих або напівструктурованих даних підходять і програмно обумовлені рішення, які до того ж мають переваги в плані забезпечення лінійної масштабованості ємності й продуктивності.

Стрімке зростання обсягів даних є серйозною проблемою для багатьох підприємств. Підтвердженням цьому може служити розвиток середньостатистичного ЦОД протягом останніх шести років. Якщо припустити, що в 2023 році обсяг даних, що зберігаються в ньому, був дорівнює 100 Тбайт, а середній щорічний приріст становив 50% (цілком звичайний показник), то до 2028 року вони повинні були перетворитися в масив обсягом більше 1 Пбайт.

Однак подібне стрімке зростання спостерігається не для всіх видів даних. Відповідно до досвіду середніх і великих підприємств, за останні п'ять-шість років помітно змінилася пропорція між структурованими й неструктурованими даними. Протягом довгого часу обидва сегменти були приблизно рівні, а сьогодні близько 90% загального обсягу збереженої інформації доводиться на частку напівструктурованих і неструктурованих даних. І саме в цій області, як правило, спостерігається експонентний ріст.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи масштабованої розподіленої файлової системи.

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

– Огляд існуючих систем масштабованої розподіленої файлової системи.

– Дослідження системи масштабованої розподіленої файлової системи.

– Програмна реалізація системи масштабованої розподіленої файлової системи.

Об'єктом дослідження є процес масштабованої розподіленої файлової системи.

Предметом дослідження є методи масштабованої розподіленої файлової системи.

Методи дослідження базуються на методах файлових систем, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод масштабованої розподіленої файлової системи.

– Розроблено вітчизняний продукт масштабованої розподіленої файлової системи, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі масштабованої розподіленої файлової системи.

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

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

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи масштабованої розподіленої файлової системи, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Адміністрування файлової системи для великого, зростаючого комп'ютера, побудованого за сучасними технологіями, є трудомістким завданням. Щоб зберігати більше файлів і обслуговувати більше користувачів, потрібно додати більше дисків, підключених до більшої кількості машин. Кожен із цих компонентів потребує людського управління. Групи файлів часто вручну призначаються певним дискам, а потім вручну переміщуються або копіюються, коли компоненти заповнюються, виходять з ладу або стають гарячими точками продуктивності. Об'єднання кількох дисків в один блок за допомогою технології RAID є лише частковим рішенням; Проблеми адміністрування все ще виникають, коли система стає достатньо великою, щоб вимагати кількох RAID і кількох серверних машин.

Нова масштабована розподілена файлова система, керує набором дисків на кількох машинах як єдиним спільним пулом сховища. Передбачається, що машини знаходяться під спільним адмініструванням і можуть безпечно спілкуватися. Були багаторічні спроби створити розподілені файлові системи, які добре масштабуються за пропускною здатністю та ємністю [1, 11, 19, 20, 21, 22, 26, 31, 33, 34]. Одна відмінна риса полягає в тому, що він має дуже просту внутрішню структуру – набір взаємодіючих машин використовує загальне сховище та синхронізує доступ до цього сховища за допомогою замків. Ця проста структура дозволяє нам виконувати відновлення системи, реконфігурацію та балансування навантаження за допомогою невеликого обладнання. Іншим ключовим аспектом є те, що він поєднує в собі різні функції, що полегшує використання та адміністрування, ніж існуючі файлові системи, про які ми знаємо:

1. Усім користувачам надається послідовний перегляд того самого набору файлів.

2. Більше серверів можна легко додати до наявної інсталяції, щоб збільшити її ємність і пропускну здатність, не змінюючи конфігурацію існуючих серверів і не перериваючи їх роботу. Сервери можна розглядати як «цеглинки», які можна поступово складати, щоб побудувати необхідну велику файлову систему.

3. Системний адміністратор може додавати нових користувачів, не піклуючись про те, які машини керуватимуть їхніми даними або на яких дисках їх зберігатимуть.

4. Системний адміністратор може створити повну та послідовну резервну копію всієї файлової системи, не виводячи її з ладу. Резервні копії можна додатково зберігати в Інтернеті, дозволяючи користувачам швидко отримувати доступ до випадково видалених файлів.

5. Файлова система терпить і відновлює після збоїв машини, мережі та диска без втручання оператора.

Система розташована поверх Petal [24], легкої в адмініструванні розподіленої системи зберігання, яка надає віртуальні диски своїм клієнтам. Як і фізичний диск, віртуальний диск Petal забезпечує сховище, яке можна читати або записувати блоками. На відміну від фізичного диска, віртуальний диск надає розріджений адресний простір розміром 264 байти, а фізичне сховище виділяється лише за вимогою. Petal додатково копіює дані для високої доступності. Petal також забезпечує ефективні знімки [7, 10] для підтримки узгодженого резервного копіювання. Frangipani успадковує більшу частину своєї масштабованості, відмовостійкості та простого адміністрування від основної системи зберігання, але для розширення цих властивостей на рівень файлової системи потрібне ретельне проектування.

1.2 Область застосування

Інформація та її безпека – це те, що пришвидшує технологічні винаходи в цю сучасну епоху – епоху інформаційних технологій. Таким чином, технологія проникла в усі сфери життя, включаючи освіту, торгівлю, армію, охорону здоров'я, право тощо. Інтернет як інструмент пропонує можливість глобального обміну інформацією та співпраці [1]. Комп'ютерне сховище стає все більш важливою частиною Інтернету; отже, забезпечення безпеки та цілісності збережених даних є надзвичайно важливою потребою. Таким чином, було помічено, що система розподіленого обміну файлами погано сприймається та/або застосовується в країнах, що розвиваються, таких як Україна, на відміну від глобальних і багатонаціональних компаній, таких як Google, які переважно розробили повну концепцію. Українські академічні установи не залишилися осторонь цього, здавалося б, основного виклику, враховуючи регулярне зростання та переміщення даних: повна автоматизація академічної діяльності, такої як розповсюдження внутрішніх записок/пошти та наукових робіт/статей у розподіленому середовищі, що належить школі. (локальна мережа) стало майже неможливим завданням через проблеми, які представляє розподілена файлова система. Однак більш уважне вивчення цього «важливого» технологічного винаходу, розподіленої файлової системи (DFS), показує, що воно створює низку проблем, які не можна ігнорувати. Хоча це вважається і стало нормальною частиною нашого повсякденного життя, воно створює проблему масштабованості (особливо, коли сервер потрібно підключити або демонтувати) і повільне транспортування файлів через трафік і безпеку. Очевидно, що атаки зловмисників та інсайдерів призвели до мільярдів найр втрачених доходів і витрачених зусиль на вирішення проблем. Більшість організацій, включно з деякими академічними установами, значною мірою покладаються на своє розподілене обчислювальне середовище, яке зазвичай складається з робочих станцій і спільної файлової системи. Ця файлова система часто зберігається на «централізованому файловому

сервері», яким керує системний адміністратор, який має доступ до всієї файлової системи, залишаючи дані вразливими для будь-кого, хто може довести (легітимність чи інше), що він є адміністратором. Нещодавно диски, підключені до мережі, також почали замінювати традиційні централізовані системи зберігання. У таких системах диски підключаються безпосередньо до мережі та покладаються на власну безпеку, а не на захист сервера. Однак таке розташування ускладнює безпеку, оскільки диск безпосередньо піддається потенційним атакам, а не прихований за одним сервером.

Крім того, існуюча система обміну файлами та її моделі контролю доступу були викликані масштабами та адміністративною складністю Інтернету. Прикладом такої системи є мережевий обмін файлами (NFS). І такі системи поділяють властивість, що відносно невелика група користувачів має доступ для читання/запису до файлів, оскільки поточні системи контролю доступу покладаються на автентифікацію, яка вимагає, щоб користувач був відомий системі [1]. Крім того, очевидно, що жоден клієнт не повинен бути стурбований або впливати на серверні проблеми (тобто збої сервера та такі дії, як монтування нового та демонтування існуючого сервера). На жаль, ця проблема на порядку денному. Після збою сервера в інтрамережі всі дії на всіх клієнтах припиняються; отже, терміново потрібне краще та довготривале вирішення цих викликів, що виникають. А масштабована розподілена система обміну файлами – це відповідь. Таким чином, у цій роботі рекомендовано використання двох або більше серверів для досягнення масштабованості розподіленої системи обміну файлами з мережевою операційною системою LINUX. Таким чином, інституційний робочий процес не буде зупинено, коли сервер вийде з ладу, оскільки існує резервний сервер, який забезпечує роботу мережевої системи/середовища.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи масштабованої розподіленої файлової системи, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Програмно-визначаємі системи Dell і Nexenta для зберігання даних

Програмно-визначаємі рішення Dell і Nexenta поєднують перевірене встаткування Dell і комплексне ПЗ Nexenta з відкритим вихідним кодом.

Рисунок 2.1 – Інтерфейс користувача програмно-визначаємого рішення Dell і Nexenta

Ці рішення допомагають забезпечити відповідність корпоративних ЦОД вимогам до ємності й продуктивності з мінімальними витратами, а також підтримують модернізацію в міру необхідності, без переривання роботи. Це

комплексний апаратного й програмний стік сервера, мережного встаткування й системи зберігання даних надає наступні переваги.

– Підтримка SAN і NAS.

– Необмежене число моментальних копій і клонування.

– Реплікація на рівні файлів і блоків.

– Потокова дедуплікація.

Розширена ємність для віртуалізованих і файлових робочих навантажень

Виберіть із сімейства рішень Dell і Nexenta конфігурацію з оптимальною ємністю, що дозволить задовольнити вимоги великомасштабних віртуалізованих і файлових робочих навантажень, забезпечивши при цьому:

– Адаптацію до непередбаченого росту й розвитку технологій .

– Переваги гнучкості конфігурування пам'яті й підключень .

– Економічне масштабування в міру зміни вимог до системи зберігання даних.

Корпоративні можливості для підвищення ефективності роботи й захисти даних

Ефективне керування й захист даних за допомогою комплексного набору функцій корпоративного класу, які допомагають спростити адміністрування систем зберігання й підвищити ефективності використання ресурсів завдяки наступним можливостям.

– Убудований стиск для видалення надлишкових даних і скорочення ємності, необхідної для зберігання блокових і файлових даних.

– Необмежене число моментальних копій для поліпшення показників цільового часу відновлення (RPO) і цільової крапки відновлення (RTO).

– Спрощені конфігурації аварійного відновлення й захист блокових і файлових даних без часу, що займає багато, резервного копіювання.

Універсальне, економічне масштабування

Нарощуйте систему й адаптуйтеся до зростаючих потреб бізнесу без додаткових витрат на ліцензування програмного забезпечення, вибираючи із широкого спектра рішень із номінальною ємністю від 44 до 960 Тбайт.

– Забезпечте відповідність вимогам до продуктивності і ємності завдяки широкому вибору процесорів, ОЗП, кешу й дискових накопичувачів.

– Розширюйте систему без додаткових витрат на програмне забезпечення й обновляйте апаратну платформу, переходячи на новітні технології для вашого бізнесу.

– Нарощуйте ємність без переривання роботи, додаючи додаткові блоки систем зберігання даних Dell.

Системи зберігання даних Dell з підтримкою технології дискових просторів Microsoft (DSMS)

Системи зберігання даних Dell з підтримкою технології дискових просторів (DSMS) допомагають знизити сукупну вартість володіння й підвищити ефективність керування життєвим циклом системи.

Скористайтеся послугами глобальної підтримки Dell по розгортанню, консалтингу, забезпеченню відповідності нормативним вимогам, а також забезпеченню необхідного рівня якості й обслуговування.

– Безперервна підтримка протягом усього життєвого циклу рішення забезпечує відчутні переваги при масштабуванні, розгортанні й експлуатації рішення.

– Рішення для зберігання даних зі змінюваними рівнями продуктивності й резервування призначено для розміщення навантажень Microsoft і характеризується спеціальними номерами SKU.

– Підтримка стандартних засобів керування включає можливість відновлення з використанням єдиного модуля ПЗ.

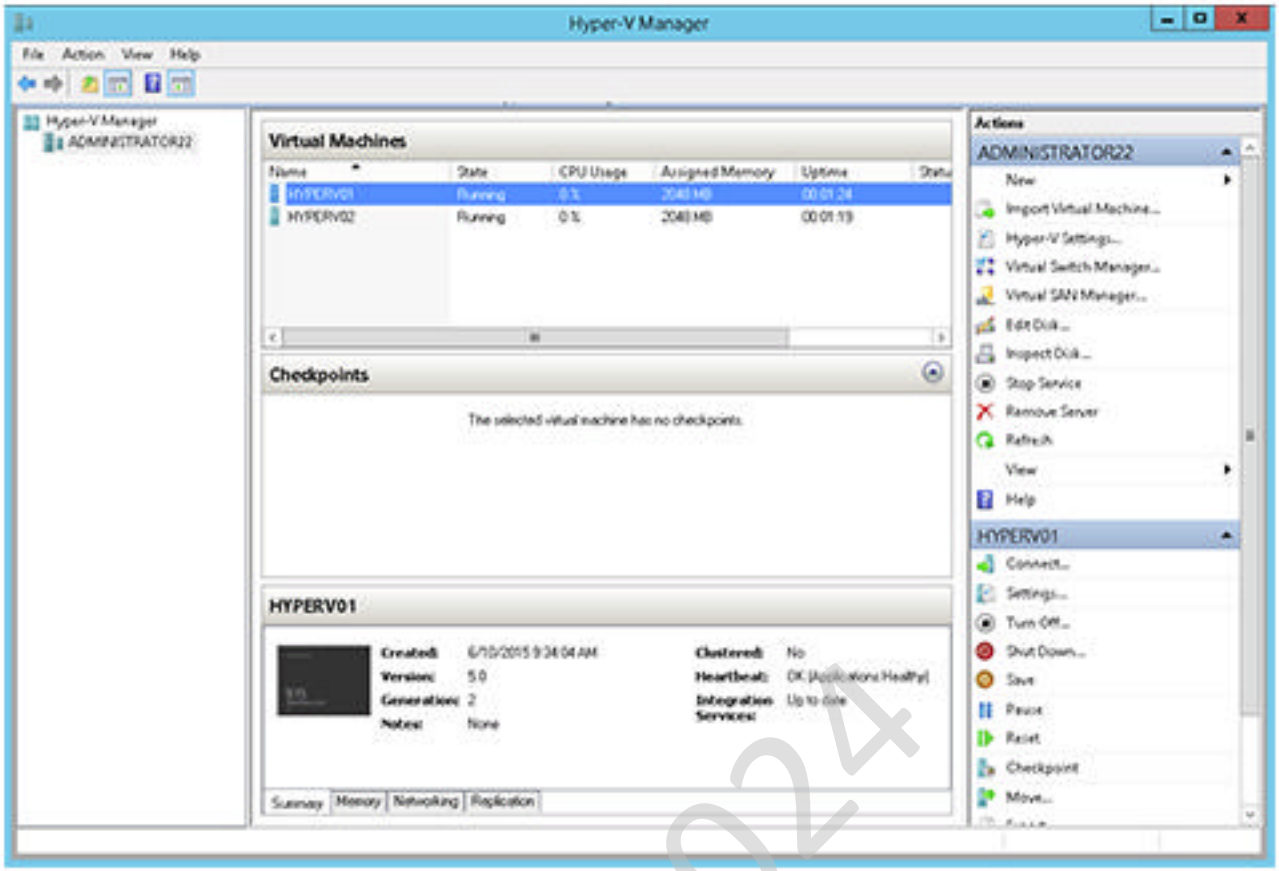

Рисунок 2.2 – Інтерфейс користувача системи зберігання даних Dell з підтримкою технології дискових просторів (DSMS)

Простота масштабування й керування

DSMS під керуванням Windows Server 2022 R2 повністю інтегрується із засобами Microsoft:

– Скорочення часу простоїв навантажень за рахунок побудови кластерної системи зберігання даних з високої відказостійкістю, заснованої на використанні загальних томів кластера й кластерних дискових просторів.

– Зниження затримок при операціях випадкового запису й мінімізація негативного впливу на швидкість передачі даних за допомогою кешу зі зворотним записом дискових просторів.

– Баланс ємності й продуктивності завдяки технологіям автоматизації системи зберігання даних на базі дискових просторів

– Підвищення доступності даних завдяки підтримці інформованості про корпуси, реалізованої в рамках технології дискових просторів.

– Можливість одержання підтримки світового рівня завдяки маршрутизації звернень до спеціалізованої черги, пов'язану із системами зберігання даних, а також допомоги при установці й розгортанні рішення завдяки послугам групи розгортання корпоративних рішень (EDT) і підрозділу професійних послуг Dell.

– Істотне зниження ризиків завдяки можливості одержати попередньо настроєні, перевірені й протестовані рішення.

Відмінний варіант для замовників з навантаженнями, що змінюються

– Система зберігання даних Dell з підтримкою технології дискових просторів Microsoft являє собою ефективний модуль для створення системи зберігання даних. Проста й ефективна архітектура рішення допомагає формувати наскрізні еталонні архітектури систем для обробки конкретних навантажень. Рішення DSMS відмінно пасують замовникам, що розгортають приватні хмари (при використанні Microsoft Azure), VDI-інфраструктури, SQL-сервери, середовища Hyper-V, а також тим замовникам, яким потрібна система зберігання для резервного копіювання й відновлення даних. Крім того, для рішень DSMS передбачені оптимізовані номери SKU для стандартних конфігурацій ємністю від декількох терабайт до більш ніж петабайта.

Конвергентний пристрій Dell EMC XC Web-Scale Converged Appliance

Конвергентні пристрої Dell EMC Web-Scale Converged Appliance серії XC на базі програмного забезпечення Nutanix дозволяють консолідувати всі обчислювальні ресурси й ресурси зберігання даних в одному корпусі. Пристрою серії XC швидко встановлюються, легко інтегруються в будь-який центр обробки даних і можуть розгортатися для декількох віртуалізованих робочих навантажень, включаючи віртуалізацію настільних систем, проекти баз даних і часток хмарних сервісів. Завдяки пристроям серії XC підприємства можуть наступне:

– Забезпечити поетапне розширення в рамках горизонтальної архітектури з оплатою в міру росту потреб.

– Реалізувати додавання окремих вузлів для збільшення ємності й продуктивності.

– Гарантувати планування майбутнього росту потреб без виділення надлишкових ресурсів.

– Скорочення витрат на IT-інфраструктуру – включаючи обслуговування, устаткування, енергію, ліцензування програмного забезпечення й сервера, сховище й мережне встаткування – до 31 %.

Спрощення й оптимізація IT-інфраструктури

Установивши у своїй інфраструктурі пристрою XC на базі Web-Scale, IT-адміністратори можуть управляти віртуальними середовищами на рівні віртуальних машин з використанням політик, заснованих на потребах кожного навантаження, замість того щоб управляти окремими логічними номерами пристроїв (LUN), томами або групами RAID. За допомогою програмного забезпечення Acropolis і Prism, що працює на пристрої, жорсткі диски й твердотільні накопичувачі можуть бути консолідовані на всіх вузлах за допомогою деталізованого подання ресурсів, автоматичної компресії й дедуплікації для збільшення ефективного використання простору сховища. Завдяки модульній і добре збалансованій архітектурі можна збільшувати продуктивність і ємність середовищ, не перериваючи їхньої роботи. Завдяки цьому середовища легко й швидко адаптуються до динамічних, постійно мінливим потребам.

На основі випробуваної технології

Базовим устаткуванням для конвергентних пристроїв серії XC є перевірена часом серверна платформа PowerEdge 13-го покоління, однак у них також реалізований цілий ряд новітніх програмних технологій, які забезпечують роботу передових інфраструктур на основі технології Web-Scale і хмарних інфраструктур. Ці пристрої з форм-фактором 1U і 2U, на які поширюється дія

послуг Dell EMC по глобальному обслуговуванню й підтримці, попередньо настроєні для віртуалізованих робочих навантажень і призначені для забезпечення готовності даних у випадку збоїти вузла й диска.

Зниження ризиків для IT-інфраструктури, підвищення адаптуємості бізнесу

Веб-масштабований пристрій XC дозволить впровадити у вашій організації технологію, що повністю змінить правила гри. Завдяки йому ви зможете розгортати нові додатки з мінімальними початковими інвестиціями й з легкістю усувати складність IT-інфраструктури, пов'язану з постійним ростом вашого бізнесу:

– Задоволення потреб у ємності й продуктивності за допомогою поетапного додавання окремих вузлів без надлишкового виділення ресурсів.

– Створення системи зберігання даних корпоративного класу завдяки програмним послугам для систем зберігання даних, таким як моментальні копії, реплікація, клонування, стиск, дедуплікація, динамічне виділення ресурсів і багато чого іншого.

– Забезпечення максимальної продуктивності за допомогою вдосконаленого розподілу даних по рівнях і локальній флеш-памяти.

– Максимальна адаптивність завдяки універсальним конфігураціям, включаючи графічний процесор і платформи скороченої глибини, а також підтримці гіпервізорів ESXi, Hyper-V і Nutanix Acropolis (AHV).

Сімейство мережних пристроїв зберігання даних (NAS) Dell Storage NX

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

– Попередньо встановлена ОС для простоти розгортання.

– Оптимізація для роботи з файлами в Windows Storage Server 2022 R2.

– Функції кластеризації при використанні NX3330.

Платформи PowerEdge

Програмно-апаратні комплекси NAS на базі технологій, використовуваних у стійкових серверах PowerEdge 13-го покоління, гарантують високу швидкість і ефективність роботи:

– Багатоядерні процесори Intel® Xeon® нового покоління

– До 32 Гбайт пам'яті й 120 Тбайт доступної ємності (NX3230)

– Варіанти накопичувачів SATA, SAS і NL-SAS

Варіант використання як шлюз мережної системи зберігання даних

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

Просте керування

Затрачайте менше часу на керування даними й більше часу на розвиток бізнесу завдяки Microsoft® System Center і вилученому робітникові столу для функції адміністрування в Windows Storage Server 2022 R2 або керуванню даними за допомогою убудованого контролера вилученого доступу Dell iDRAC7 Enterprise Edition. Крім того, ви можете підвищити ефективність використання ємності мережної системи зберігання даних і оптимізувати дисковий простір за допомогою наступних убудованих функцій пристроїв сімейства Dell Storage NX:

– Дедуплікація даних.

– Деталізоване надання ресурсів.

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

– Служба тіньового копіювання томів (VSS) для загальних файлових папок SMB, що дозволяє створювати резервні копії операцій з використанням

моментальних знімків вилучених загальних файлових папок, що підтримують серверні додатки на основі SMB.

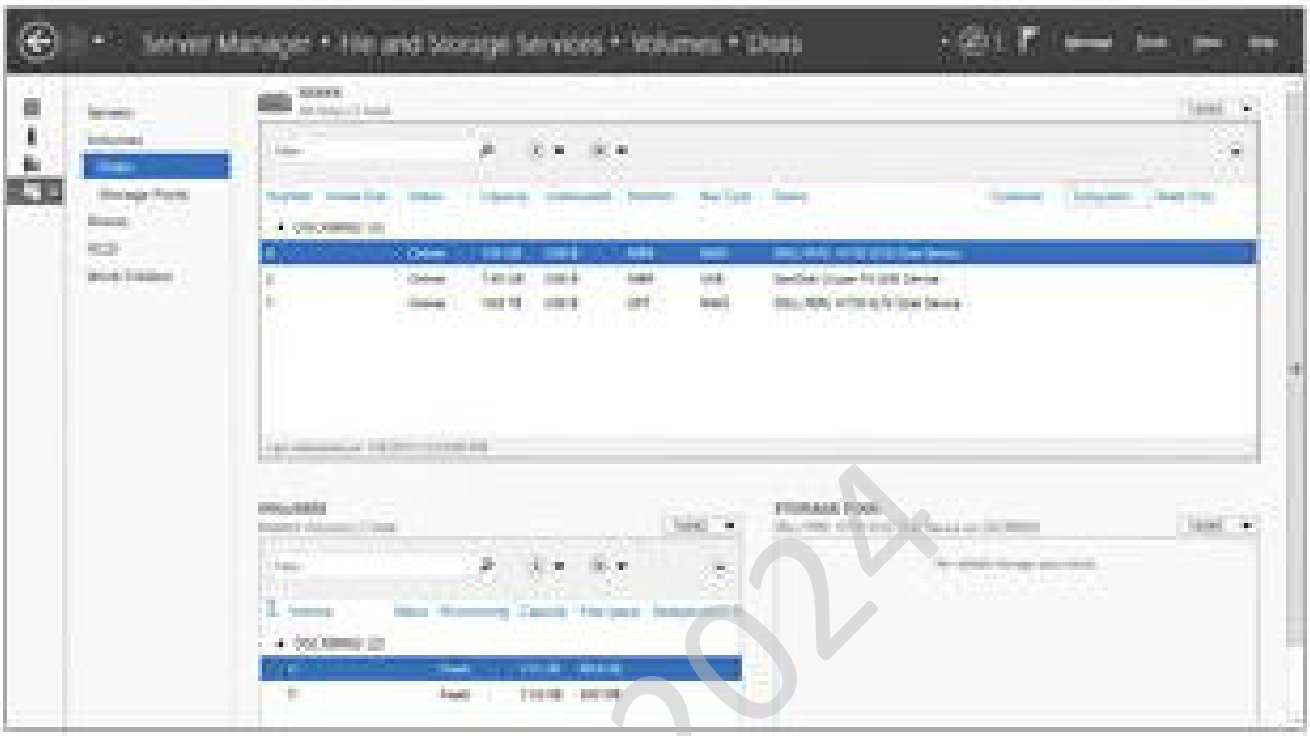

Рисунок 2.3 – Інтерфейс користувача пристрою зберігання даних Dell Storage NX

Оновлені ОС

Сімейство Dell Storage NX працює на базі програмного забезпечення Windows® Storage Server 2022 R2, що забезпечує продуктивність і багатофункціональність Microsoft Windows Server® 2022 R2 у роботі пристроїв NAS. Це наступні продукти:

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

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

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

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# 2022 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.

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

Взаємодія між мовами є ключовою особливістю .NET Framework. Оскільки код IL, створюваний компілятором Visual C# відповідає специфікації CTS, код IL на основі Visual C# може взаємодіяти з кодом, створюваним версіями мов Visual Basic, Visual C++, Visual J# платформи .NET Framework і ще більш ніж

20 CTS-сумісних мов. В одному складанні може бути кілька модулів, написаних на різних мовах платформи .NET Framework, і типи можуть посилатися один на одного, як якби вони були написані на одній мові.

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

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи масштабованої розподіленої файлової системи.

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

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

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

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

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації

в роботі технологічного обладнання;

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Обмін файлами може означати розподіл доступу до фізичних або електронних файлів. Хорошим прикладом є передача документів або файлів певному персоналу організації для спільної мети вручну або на пристрої зберігання, такі як флеш-диск, компакт-диск або жорсткий диск.

Однак спільний доступ до файлів майже завжди означає спільний доступ до файлів у мережі, навіть якщо вона знаходиться в невеликій локальній мережі (LAN) [2]. Це практика розповсюдження або надання доступу до цифрових медіа, таких як комп’ютерні програми, мультимедіа (аудіо, зображення та відео), документи або електронні книги [3]. Це публічний або приватний спільний доступ до комп'ютерних даних або простору в мережі з різними рівнями привілеїв доступу [2]. Отже, спільний доступ до файлів включає дві або більше осіб (точніше кажучи, комп’ютери) для доступу або використання файлу для читання чи запису, або обох. Таким чином, він підкреслює практику обміну або надання доступу до цифрової інформації чи ресурсів, включаючи документи, мультимедіа (аудіо/відео), графіку, комп’ютерні програми, зображення та електронні книги. Це приватний або публічний розподіл даних або ресурсів у мережі з різними рівнями привілеїв спільного використання.

Розподілена файлова система (DFS) підтримує обмін інформацією у формі файлів по всій інтрамережі. І це дозволяє користувачам зберігати та отримувати віддалені файли як локально, але з будь-якого комп’ютера в інтрамережі. Таким чином, це програма на основі клієнта/сервера, яка дозволяє клієнтам отримувати доступ і обробляти дані, що зберігаються на сервері, як на їх власному комп’ютері [4] [5]. Коли користувач отримує доступ до файлу на сервері, сервер надсилає користувачеві копію файлу, яка зберігається в кеші на комп’ютері

користувача під час обробки даних, а потім повертається на сервер. В ідеалі розподілена файлова система організовує служби файлів і каталогів окремих серверів у глобальний каталог таким чином, щоб віддалений доступ до даних не залежав від місця розташування, а був ідентичним для будь-якого клієнта. І всі файли стають доступними для всіх користувачів глобальної файлової системи, а організація є ієрархічною та базується на каталозі. Він пропонує нам файлову систему, яка зберігає свої дані на сервері, і до її даних можна отримати доступ і використовувати їх так, ніби вони знаходяться на локальному вузлі. Розподілена файлова система дозволяє зручно обмінюватися інформацією та файлами між користувачами в мережі контрольованим і авторизованим способом. А сервер дозволяє користувачам клієнта обмінюватися файлами та зберігати дані так само, як вони зберігають інформацію локально. Однак сервер має повний контроль над даними та надає контроль доступу клієнтам.

Фактори, що впливають на продуктивність розподіленої файлової системи порівняно з традиційною системою клієнт/сервер

Мета розподіленої файлової системи (DFS) полягає в тому, щоб дозволити користувачам фізично розподілених комп'ютерів спільно використовувати дані та ресурси зберігання за допомогою спільної файлової системи [5]. І, на відміну від традиційних рішень клієнт/сервер, у розподіленій файловій системі можна досягти кращої продуктивності. Фактори, які впливають на кращу продуктивність розподіленої файлової системи, включають:

1. Замість того, щоб зберігати дані на одному сервері, дані можна зберігати на кількох вузлах. Це відомо як реплікація.

2. Система завжди доступна щоразу, коли до неї підключається клієнт. Це надійність.

3).Система має можливість обслуговувати запити клієнтів при кожному вході в екземпляр. Це називається доступністю.

4. Дані не втрачаються, але захищені, оскільки вони не зберігаються на одному сервері.

У порівнянні з традиційною системою клієнт/сервер, де дані зберігаються на одному сервері, унікальна функція безпеки розподіленої файлової системи полягає в тому, що важливі або часто потрібні дані в розподіленій файловій системі можуть зберігатися на кількох вузлах (вузли означає комп'ютер, що працює в розподілена файлова система) [6]. Це називається «реплікацією». Реплікацію можна використовувати для досягнення кращої продуктивності системи та/або «надійності» системи. Дані в розподіленій файловій системі більш захищені від збою вузла. Якщо один або кілька вузлів виходять з ладу, інші вузли можуть забезпечити всі функції. Ця властивість також відома як «доступність» або «надійність». Різниця між доступністю та надійністю проста: доступність означає, що система може обслуговувати запит клієнтів у момент, коли клієнт підключається до системи. Надійність означає, що система доступна весь час, коли до неї підключаються клієнти. Файли також можна переміщувати між вузлами. Зазвичай це викликає адміністратор, і це робиться для покращення балансування навантаження між вузлами. Користувачі не повинні знати, де розташовані служби, а також передача з локальної машини на віддалену також має бути прозорою. У розподіленій файловій системі ця властивість відома як прозорість. Якщо ємності вузлів недостатньо для зберігання файлів, до існуючої розподіленої файлової системи можна додати нові вузли, щоб збільшити її ємність. І це відомо як «масштабованість». Зазвичай клієнт зв'язується з розподіленою файловою системою за допомогою локальної мережі (LAN).

Ключові характеристики розподіленої файлової системи

Очікується, що розподілена файлова система матиме три основні функції, які забезпечать надійне та захищене середовище обміну файлами серед багатьох інших; а саме прозорість, відмовостійкість і масштабованість.

1. Прозорість: користувачі повинні отримувати доступ до системи незалежно від місця входу, мати можливість виконувати однакові операції з розподіленою файловою системою та локальною файловою системою, і не повинні піклуватися про помилки через розподілену природу файлової системи;

завдяки механізмам відмовостійкості. Крім того, Transparency можна розглядати з точки зору продуктивності. Таким чином, маніпуляції з даними мають бути принаймні такими ж ефективними, як у звичайних файлових системах. Іншими словами, складність основної системи має бути прихована від користувачів: кінцевому користувачеві не потрібно знати, як спроектована система, як дані мають бути розташовані та доступні, і як виявляються несправності [7]. І функції, які забезпечують прозорість:

– Іменування: це зіставлення між локальною назвою та фізичним розташуванням даних. Наприклад, у класичній файловій системі клієнти використовують логічне ім’я (текстове ім’я) для доступу до файлу, який відображається на фізичних блоках диска. А в розподіленій файловій системі потрібно додати імена серверів, що містять диск, на якому зберігаються дані. Розподілена файлова система повинна дотримуватися прозорості розташування: деталі того, як і де зберігаються файли, приховані для клієнтів. Крім того, може існувати кілька копій файлів, тому зіставлення має повертати набір розташувань усіх доступних копій. Розподілена файлова система має бути незалежною від розташування: логічне ім’я не повинно змінюватися, навіть якщо файл переміщується в інше фізичне розташування. Для цього використовуються таблиці розподілу або складні алгоритми для забезпечення глобальної структури простору імен, яка є однаковим простором імен для всіх клієнтів.

– Інтерфейси доступу до даних: клієнти можуть створювати, читати, писати, видаляти файли, не замислюючись про складні механізми основної системи, яка виконує операції, і має бути забезпечений доступ до системи за допомогою простих інструментів. Ось кілька прикладів таких інструментів:

i) Інтерфейс командного рядка (CLI), який використовується для доступу до файлів за допомогою традиційних команд UNIX.

ii) Java, CC $^{++}$, інші мови програмування та REST (веб-інтерфейс) API можна використовувати для графічного інтерфейсу, такого як провідник вікон.

iii) Користувачам можна дозволити монтувати (приєднувати) віддалені

каталоги до їх локальної файлової системи: таким чином, доступ до віддалених файлів так, ніби вони зберігаються на локальному пристрої. Деякі приклади механізму FUSE або команди монтування UNIX.

– Кешування: це техніка, яка полягає в тимчасовому зберіганні запитуваних даних у пам’яті клієнта. Розподілені файлові системи використовують кешування, щоб уникнути додаткового мережевого трафіку та споживання ЦП, викликаного повторними запитами до того самого файлу, і таким чином підвищити продуктивність. Коли дані потрібні вперше, з сервера, який зберігає їх, робиться копія в основну пам’ять клієнта. Таким чином, для кожного наступного запиту цих даних клієнт буде використовувати локальну копію, уникаючи зв’язку з сервером і доступу до диска. Ця функція пов’язана з прозорістю продуктивності, оскільки запити можуть виконуватися швидко, приховуючи передачу даних користувачам за допомогою цієї техніки. Однак, коли дані змінюються, модифікація повинна бути передана серверу та будь-якому іншому клієнту, який кешував дані [7].

Таким чином, кеш-пам’ять у комп’ютерній системі розглядається як компонент, який зберігає запитувані дані, отже, може потенційно використовуватися в майбутньому [6]. Коли запитуються кешовані дані, час відповіді коротший, ніж коли дані не зберігаються в кеші та їх потрібно завантажити. Кешовані дані можна зберігати в оперативній пам’яті для швидкого доступу та/або на жорсткому диску. Кеш-пам'ять може бути з обох сторін зв'язку. На стороні сервера кеш зазвичай знаходиться в оперативній пам'яті. На стороні клієнта кеш може розташовуватися в оперативній пам'яті або на жорсткому диску. На стороні сервера, якщо вміст файлу кешується, механізм кешу економить час, оскільки немає необхідності отримувати доступ до вмісту файлу з жорсткого диска. На стороні клієнта, якщо клієнт запитує файл, механізми кешу економлять час, оскільки немає необхідності спілкуватися з сервером. Кешування на стороні клієнта також іноді називають реплікацією, ініційованою клієнтом. Клієнтський кеш також може надавати так званий офлайн-кеш. Це означає, що

клієнт може отримати доступ до файлу з кешу після відключення від сервера. Автономний кеш часто зберігається на жорсткому диску; Механізм автономного кешування використовується в CODA.

– Виявлення несправностей: відмовостійку систему не слід зупиняти у разі тимчасових або часткових відмов. Розглянуті помилки – це збої мережі та сервера, які роблять служби даних недоступними, цілісність і узгодженість даних, коли кілька користувачів одночасно отримують доступ до даних. Іншими словами, це здатність виявляти перевантажені сервери, виправляти поведінку сервера або пошкоджені дані та приймати рішення щодо усунення цих несправностей. У розподіленій файловій системі помилки повинні бути виявлені системою з використанням мінімальних ресурсів перед виправленням, щоб користувачі не знали, що такі помилки виникають. Усі машини спілкуються разом у прозорий спосіб, обмінюючись невеликими повідомленнями. Наприклад, звіт дозволяє серверам, що керують простором імен, знати, які дані зберігаються на якому сервері. Оскільки дані постійно переміщуються, це дозволяє системі визначити, які дані втрачено та потребують переміщення або повторного копіювання, коли сервер стає недоступним або перевантаженим. Ще одне повідомлення – це серцеві сигнали, які використовуються для підтвердження доступності сервера. Якщо якийсь час не надсилає серцеві сигнали, він переміщується в карантин, а звітні повідомлення використовуються для застосування правильного рішення [7].

2. Стійкість до відмов: помилками вважаються збої мережі та сервера, які роблять дані та служби недоступними, цілісність і узгодженість даних, коли кілька користувачів одночасно отримують доступ до даних. Тому відмовостійку систему не слід зупиняти у разі цих тимчасових або часткових збоїв: у розподілених файлових системах збій мережі та сервера є нормою, а не винятком. Інструменти повинні бути розгорнуті для підтримки та забезпечення постійної доступності даних, щоб гарантувати обробку запитів у разі помилок. Необхідно також враховувати цілісність і узгодженість даних, оскільки передбачені такі

механізми, як кешування або реплікація.

Крім того, деякі функції для забезпечення відмовостійкості:

– Політика реплікації та розміщення: для того, щоб дані були завжди доступними, навіть якщо сервер виходить з ладу, розподілені файлові системи використовують реплікацію файлів шляхом створення кількох копій даних на різних серверах. Коли клієнт запитує дані, він прозоро отримує доступ до однієї з копій. Щоб підвищити відмовостійкість, репліки зберігаються на різних серверах відповідно до політики розміщення. Наприклад, репліки можна зберігати на різних вузлах, на різних доріжках, у різних географічних місцях, так що, якщо в будь-якому місці системи виникне збій, дані залишаться доступними.

– Синхронізація: у розподілених файлових системах необхідно враховувати синхронізацію між копіями даних. Коли дані перезаписуються, усі їх копії необхідно оновити, щоб надати користувачам останню версію даних. Існує три основні підходи:

і) У синхронному методі будь-який запит щодо змінених даних блокується, доки не буде оновлено всі копії. Це забезпечує користувачам доступ до останньої версії даних, але затримує виконання запитів.

іі) У другому методі, який називається асинхронним, запити щодо змінених даних дозволені, навіть якщо копії не оновлені. Таким чином, запити можуть бути виконані в розумний час, але користувачі зможуть отримати доступ до застарілої копії.

ііі) Останній підхід є компромісом між першими двома в напівасинхронному методі, запити блокуються, доки деякі копії, але не всі, будуть оновлені. Наприклад, припустімо, що є файлові копії даних, запит щодо цих даних буде дозволено після оновлення трьох копій. Це обмежує можливість доступу до застарілих даних, одночасно зменшуючи затримку для виконання запитів.

– Узгодженість кешу: це та ж проблема, що й синхронізація – як оновити всі копії даних у кеші, коли одна з них змінена. Дані можна кешувати для

покращення продуктивності системи, що може призвести до неузгодженості між копіями, коли користувач змінює одну з них. Ці зміни потрібно поширити на всі копії та дані в кеші, щоб надати користувачам їх актуальну версію. Щоб уникнути цієї проблеми, використовуються різні підходи:

і) Лише запис/читання багатьох (WORM): це перший підхід для забезпечення лише узгодженості. Створений файл не можна змінювати. Кешовані файли знаходяться в режимі лише для читання. Тому кожне зчитування відображає останню версію даних.

іі) Другим методом є транзакційне блокування, яке полягає в отриманні блокування читання даних запиту, щоб будь-який інший користувач не міг виконати або записати дані, або оновити блокування запису, щоб запобігти будь-якому читанню або запису на диск. Таким чином, кожне читання відображає останній запис, і кожен запис виконується по порядку.

ііі) Іншим підходом є лізинг: це контракт на обмежений період між сервером, на якому зберігаються дані, та клієнтом, який запитує ці дані для запису. Оренда надається, коли запитуються дані, і під час оренди клієнту гарантується, що жоден інший користувач не зможе змінити дані. Дані знову доступні після закінчення терміну оренди або якщо клієнт відмовляється від свого права. Для майбутніх запитів на читання кеш оновлюється, якщо дані були змінені. Для майбутніх запитів на запис клієнту надається оренда, якщо це дозволено (тобто для цих даних не існує оренди або права звільняються).

– Балансування навантаження: це можливість автоматичного балансування системи після додавання або видалення серверів. Повинні бути надані інструменти для відновлення втрачених даних, їх зберігання на інших серверах або переміщення з головного пристрою на щойно доданий. Зв'язок між машинами дозволяє системі виявляти збої сервера та його перевантаження. І щоб виправити ці помилки, сервери можна додавати або видаляти. Коли сервер видаляється із системи, останній повинен мати можливість відновити втрачені дані та зберігати їх на інших серверах. Коли сервер додається до системи,

необхідно надати інструменти для переміщення даних із хост-сервера на щойно доданий сервер. Користувачі не повинні знати про цей механізм. Зазвичай розподілені файлові системи використовують запланований список, до якого вони розміщують дані для переміщення або повторного копіювання. Періодично алгоритм повторює цей список і виконує належну дію. Наприклад, CEPH використовує функцію «Контрольована реплікація під масштабованим хешуванням» (CRUSH) для випадкового зберігання нових даних, переміщення об'єкта існуючих даних до нових ресурсів зберігання та рівномірного відновлення даних із видалених ресурсів зберігання.

3. Масштабованість: це здатність ефективно використовувати велику кількість серверів, які динамічно та постійно додаються в систему [7]. Всупереч загальним відомостям про розподілену файлову систему, це означає, що ця система може залучати більше ніж один сервер у локальній мережі для кращої продуктивності спільного використання файлів. Практичним прикладом є ситуація, коли два або більше серверів використовуються в мережевому середовищі: коли один сервер у внутрішній мережі не працює, очікується, що операції та служби розподіленої файлової системи не припиняться, а синхронізуються та віртуально передаються на інший сервер. (s); отже, масштабованість. Таким чином, масштабована система розподіленого обміну файлами стала можливою завдяки децентралізації файлів сервера в інтрамережі, щоб вони не зберігалися лише на одному сервері.

Реалізація масштабованості в системі розподіленого обміну файлами

Як зазначалося раніше щодо більш детального вивчення цього відповідного технологічного рішення, розподілена файлова система (DFS) показує, що вона представляє низку проблем, які не можна ігнорувати. І ключовою проблемою, яку він представляє, є масштабованість (особливо, коли сервер потрібно підключити або відключити). Таким чином, додаткові фактори, які слід враховувати при реалізації масштабованості, включають:

1. Кількість серверів: пам'ятайте, що масштабованість передбачає, що в

розподіленому середовищі існує більше однієї серверної системи, і жоден клієнт не повинен бути недоступним, коли сервер у мережі виходить з ладу, оскільки інші сервери все ще працюють і файли копіюються на кожен з них. Таким чином, кількість серверів забезпечує і підтримує доступність і надійність внутрішньої мережі.

2. Операційна система: серед інших мережевих операційних систем рекомендується LINUX, враховуючи її надійність і захист від вірусів.

Характеристики масштабованої розподіленої файлової системи

Масштабована система клієнт/сервер має такі характеристики

1. Клієнти більше не страждають від внутрішніх проблем, таких як збій сервера.

2. До зовнішньої програми можна отримати доступ на будь-якій комп'ютерній системі в мережі; отже, подолаючи проблему небажаного та неавторизованого доступу.

3. Безпека мережевих (або організаційних) даних гарантується.

4. Клієнти зможуть легко передавати свої файли в мережевому середовищі.

Серверні механізми розроблені для зв'язку між усіма комп'ютерами в мережі; отже, пропонування та забезпечення прозорості.

З огляду на швидке зростання завдань/робочого навантаження та робочих процесів у наших сучасних установах, а також проблеми, які постають перед існуючими системами обміну файлами та їхніми моделями контролю доступу в Інтернеті, існує потреба у файловій системі з можливістю масштабування. У наші дні великі установи бажають мати автоматизований робочий процес, який допоміг би усунути виснажливі ручні процеси спілкування та обміну файлами/поштою. Таким чином, потенційним кандидатом є масштабована розподілена система обміну файлами. Серед усіх файлових систем мережева файлова система (NFS) є найбільш перевіреною технікою. У цій роботі кожен користувач/клієнт у мережі може отримати доступ до мережевої системи

організації так, ніби файли знаходяться в його локальній системі. Крім того, для досягнення масштабованості розподіленої файлової системи можна використовувати два або більше серверів. Таким чином, інституційний робочий процес не припиняється, коли сервер виходить з ладу, оскільки існує резервний сервер, який забезпечує роботу мережевої системи/середовища.

Наразі ця стаття досягла поставленої мети, яка полягає в тому, щоб запропонувати реалізацію функції «масштабованості» розподіленої системи спільного використання файлів як вирішення проблем розподіленої файлової системи (DFS), згаданих раніше.

У цій роботі представлено перехід від традиційної централізованої системи зберігання та поганої файлової системи (мережевих дисків) до масштабованої розподіленої файлової системи. Проблему масштабованості, особливо коли сервер потрібно монтувати або демонтувати, і повільне транспортування файлів можна усунути за допомогою впровадження децентралізованої серверної системи за допомогою двох або більше серверів, які працюють синхронно. А коли масштабованість буде досягнута, клієнти більше не будуть страждати від внутрішніх проблем, таких як збій сервера. Система також гарантує легшу/швидшу транспортування файлів між клієнтськими системами.

3.2 Розробка структурної схеми

Багато додатків працюють швидше й ефективніше, якщо використовується програмно-визначаєме сховище (SDS), що відокремлює програмне забезпечення від устаткування систем зберігання. Такі рішення універсальні, сприяють активізації інноваційних розробок і дозволяють зробити крок у майбутнє, адаптуючи архітектуру сховища до потреб робочих навантажень.

Деякі програмно-визначаємі підходи ставлять вас у залежність від спеціалізованого ПЗ або неадаптуємих архітектур. Корпорація Dell, навпроти,

пропонує широкий вибір сучасних SDS-Рішень і дозволяє вам використовувати встаткування й ПЗ, що підходить для ваших конкретних додатків. Завдяки цьому ви одержуєте розширений контроль над ресурсами, а прискорений доступ до апаратних і програмних інновацій дає вам можливість швидко впроваджувати нові IT-можливості**.**

Таких програмно-визначаємих рішень зберігання, які підходили б для будь-яких завдань, просто не існує. Тому, працюючи в партнерстві із провідними постачальниками ПЗ для систем зберігання, ми розробили адаптуємі відкриті рішення. Вони дозволяють зберегти існуючі капіталовкладення й розширити можливості ваших систем у майбутньому. Вибирайте з них ті, які підійдуть саме вам: ми пропонуємо як випробуваної й перевірені еталонні архітектури з мінімальним ступенем ризику, так і готові й зручні гіперконвергентні рішення, розгортання яких займає менш чверті години**.**

Швидше за все, програмно-визначаємі системи – це лише одна з тих інновацій, які ви плануєте для свого сховища. Наші платформи є по-справжньому програмно-визначаємими, тобто момент впровадження інновацій в організації визначаєте ви, а не наступний цикл відновлення встаткування. Розвивайтеся тоді, коли будете впевнені, що готово зробити крок уперед. Наприклад, серія Dell XC і розроблені Dell рішення для EVO:RAIL на базі процесорів Intel® Xeon® дозволяють впроваджувати нові функції в міру їхньої появи й розгортати нові вузли менш чим за хвилину.

Програмно-визначаємі сховища допомагають підвищити продуктивність і швидкість роботи за рахунок переносу функцій з устаткування масивів зберігання даних ближче до обчислювальних ресурсів. Інтеграція й інструменти автоматизації дозволяють зменшити складність адміністрування й заощадити робочий час. Що ви одержуєте в результаті? Інфраструктуру віртуальних робочих столів, розгортання якої займає в 6 разів менше часу при на 27% більше низькій сукупній вартості володіння. У порівнянні із традиційними підходами до сховища для інфраструктури віртуальних робочих столів, рішення здатні

підтримувати подвоєна кількість користувачів, а місця при цьому займати на 91% менше.

Хоча підтримка будь-якого встаткування й кожного ПЗ вважається головною перевагою програмно-визначаємих систем, у реальності всі набагато складніше. Рішення власного складання типу "білий ящик" часто не виправдують очікувань. Установка програмно-визначаємого сховища на випробуване встаткування корпоративного класу з підходящими конфігураціями дозволяє домогтися значно кращих результатів. Крім того, використовувати корпоративні сховища такого типу стає просто, як ніколи раніше, адже послуги підтримки світового класу дозволяють звертатися до фахівців з будь-яких питань із будь-якої крапки земної кулі.

Щоб упоратися з напливом неструктурованим і напівструктурованим даних, підприємствам потрібно масштабована життєздатна програмно обумовлена інфраструктура зберігання даних, здатна забезпечити високу доступність, економічну ефективність і простоту керування. Основою такого рішення служить добре масштабована розподілена файлова система, що при необхідності дозволяє здійснити лінійне розширення ємності ресурсів зберігання.

Лінійна масштабованість

Коли в контексті систем зберігання даних мова заходить про можливість їхнього розширення, як правило, мається на увазі так звана лінійна масштабованість – термін, що часто трактується неправильно. У теорії мається на увазі, що дворазове збільшення ємності зберігання забезпечує подвоєння продуктивності: при збереженні часу відгуку (при інших незмінних умовах), пропускна здатність (вимірювана в гігабайтах у секунду) повинна збільшитися теж у два рази. Однак на практиці традиційні системи зберігання не здатні виконати ця вимога.

Причина такої розбіжності криється в тім, що масштабованість систем зберігання залежить від безлічі факторів, і ємність сховищ – це лише один з них. Апаратне забезпечення системи зберігання, відповідальної за керування

окремими жорсткими дисками, також повинне масштабуватися відповідним чином. Для оптимального використання всіх шпинделів жорстких дисків при пікових навантаженнях потрібний центральний процесор з достатньою продуктивністю. Крім цього, підтримка подвоєної ємності зберігання можлива тільки в тому випадку, коли файлова система й функція метаданих масштабуються лінійно – тоді система зможе встановити місцезнаходження даних на нових доданих дисках. Інакше кажучи, для забезпечення лінійної масштабованості систем зберігання даних необхідно, щоб продуктивність всіх компонентів збільшувалася пропорційно.

Для традиційних розподілених систем зберігання даних це означає, що кожному вузлу зберігання доводиться нести додаткові системні витрати на комунікацію з іншими вузлами при виконанні яких-небудь операцій з файлами. Оскільки ємності збільшуються швидше, ніж продуктивність, лінійного росту потужності системи досягти не вдається.

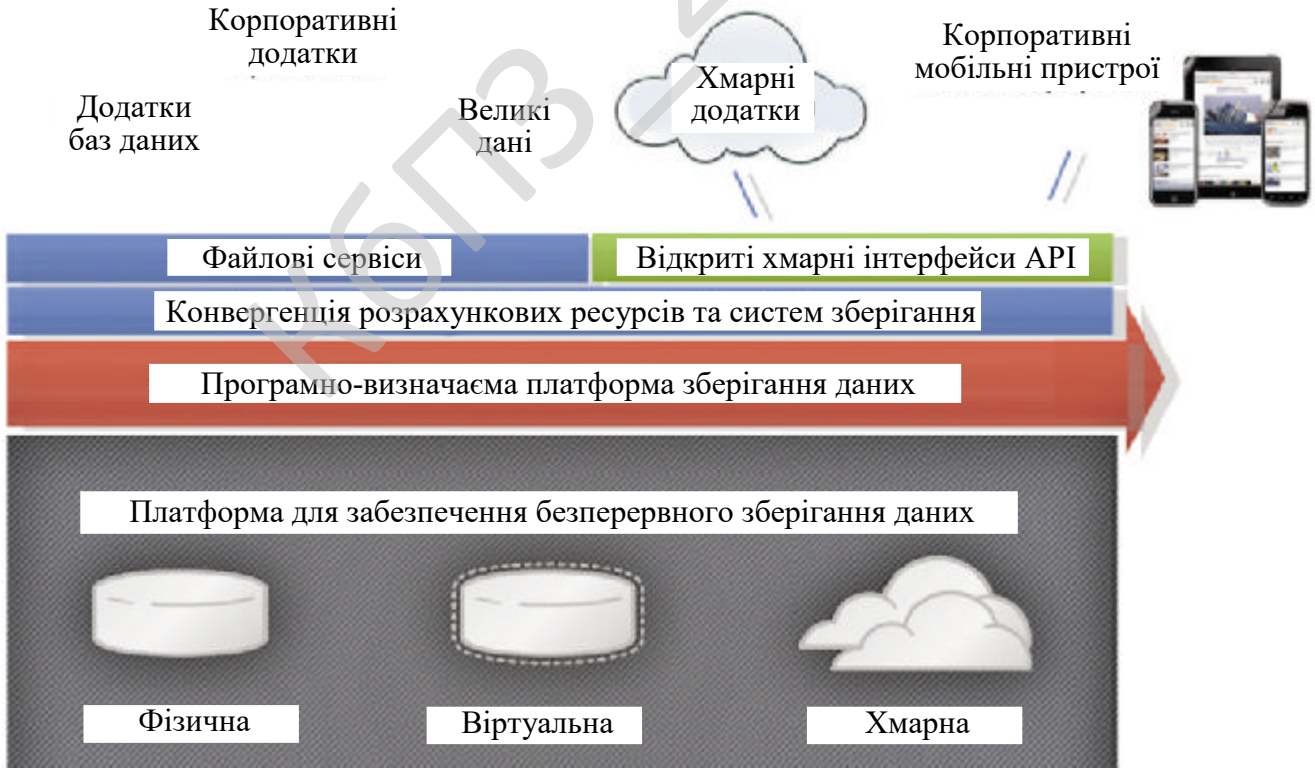

Рисунок 3.1 – Структурна схема системи

У випадку повністю програмно обумовлених масштабованих рішень зберігання даних ці обмеження не виникають – обсяги сховищ ростуть синхронно із продуктивністю (див. Рисунок 3.1). У результаті створюється ефективна інфраструктура зберігання для розміщення напівструктурованих і неструктурованих даних на стандартних серверах x86. Завдяки спільному використанню процесорів і ресурсів уведення-виводу недорогих стандартних серверів формується великий високопродуктивний пул (кластер) зберігання даних. Тепер, якщо підприємству потрібно збільшити ємність зберігання, IT-фахівці можуть просто додати додаткові жорсткі диски, а для підвищення продуктивності – збільшити число серверів. Додаткових накладних витрат при цьому не виникає.

Програмно обумовлена інфраструктура

Для лінійного збільшення продуктивності і ємності таких рішень повинні дотримуватися три умови:

– відсутність сервера метаданих;

– ефективний розподіл даних, що зберігаються, для забезпечення високого ступеня масштабованості й надійності;

– паралельний доступ до даних для досягнення максимальної продуктивності в повністю розподіленій архітектурі.

У програмно обумовлених масштабованих рішеннях логічна й фізична локалізація даних є складним завданням для розроблювачів. У більшості розподілених систем ця проблема вирішується за допомогою окремого індексу, де втримуються імена файлів і метадані для їхньої локалізації. Але в результаті виникає критична крапка відмови (Single Point of Failure) і створюється вузьке місце для продуктивності: при росту кількості серверів, жорстких дисків і даних сервер метаданих починає гальмувати роботу всього рішення. Ситуація ще більше загострюється у випадку безлічі малих по розмірі файлів, тому що обсяг їх метаданих росте випереджальними темпами. У системі компанії Red Hat, приміром, це завдання вирішується за допомогою алгоритму хешування: для

кожного ім'я файлу розраховується його хеш-сума. Це дозволяє усунути основне джерело зниження продуктивності операцій введення-виводу або навіть потенційну причину виникнення збоїв.

Одночасна підтримка зберігання файлів і об'єктів (File/Object Storage) в одному пулі зберігання також виявляється дуже корисною. Сполучення обох видів зберігання значно спрощує процес керування різними даними й забезпечує підприємствам більшу гнучкість при зберіганні корпоративної інформації, чим специфічні рішення SAN від різних виробників. Це досить вигідний спосіб протидії стрімкому зростанню обсягів напівструктурованих і неструктурованих даних.

Таким чином, програмно обумовлені рішення добре підходять для збереження різних структурованих даних, керування складним мультимедійним умістом і архівування в безпосередній близькості від робочого місця. Приміром, Posix-сумісне рішення від Red Hat підтримує такі стандарти NAS, як NFS і SMB, для забезпечення доступу до файлів і OpenStack Swift для доступу до об'єктів, а крім того, воно оснащено клієнтом Glusterfs для паралельного доступу.

Відсутність залежності від масивів зберігання даних

Якщо на передній план виходять напівструктурованого й неструктурованого дані, підприємства, що використовують повністю програмно обумовлене рішення, перестають залежати від дорогих і погано піддаються масштабуванню монолітних масивів зберігання даних. Таке рішення дозволяє в найкоротший термін уводити до ладу додаткові недорогі сервери x86 і додавати в інфраструктуру зберігання даних – будь те власний ЦОД підприємства або гібридна хмара – гарно масштабовану й високопродуктивну ємність. Синхронна реплікація даних підтримує їх локальне дзеркалювання й сприяє забезпеченню безперервності бізнес-процесів. Асинхронна реплікація даних, у свою чергу, дозволяє робити дистанційне копіювання даних для створення резервних копій на випадок аварійного відновлення.

Якщо підприємствам доводиться боротися з лавиноподібним ростом обсягів напівструктурованих і неструктурованих даних, то програмно обумовлене рішення дозволить одержати додаткову ємність із хмари. Так, інструмент керування Red Hat Storage надає адміністраторам централізований огляд усього пула зберігання даних. Він базується на проекті Ovirt – відкритій платформі для керування інфраструктурою й віртуалізацією. Завдяки цьому адміністрування зростаючих обсягів інформації спрощується без необхідності інвестування засобів у нове апаратне забезпечення, що дозволяє впоратися з підривним збільшенням кількості даних у діапазоні від декількох тера- до багатьох петабайтів.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.2. Для підвищення надійності зберігання даних у масштабованій розподіленій файловій системі у даній роботі пропонується використовувати пропонується використовувати перешкодостійке кодування, а саме циклічні коди. Одним із класів циклічних кодів, здатних виправляти багаторазові помилки, є коди БЧХ.

Примітивним кодом БЧХ, що виправляє t_u помилок, називається код довжиною $n = q^m - 1$ над GF(q), для якого елементи:

$$\alpha^1, \alpha^2, \ldots, \alpha^{2t_u},$$

є корінням багаточлена, що породжує.

Тут α – примітивний елемент GF(q^m).

Багаточлен, що породжує, визначається з вираження:

$$g(x) = НОК\left[f_1(x), f_2(x), \ldots, f_{2t_u}(x)\right],$$

де $f_1(x), f_2(x)$...– мінімальні багаточлени корінь $g(x)$.

Число перевірочних елементів коду БЧХ задовольняє співвідношенню:

$$r = n - k \le mt_u.$$

Визначити значення багаточлена, що породжує, для побудови примітивного коду над GF(2) довжини 31, що виправляє дві кратні помилки ($t_u = 2$).

Виходячи з визначення коду БЧХ коріннями багаточлена $g(x)$ є:

$$\alpha^1, \alpha^2, \alpha^3, \alpha^{4(2t_u)},$$

де α – примітивний елемент GF(q^m) = GF(2^5).

Багаточлен, що породжує, визначається з вираження:

$$g(x) = НОК[f_1(x), f_2(x), f_3(x), f_4(x)],$$

де $f_1(x), f_2(x), f_3(x), f_4(x)$ – мінімальні багаточлени корінь відповідно:

$$\alpha^1, \alpha^2, \alpha^3, \alpha^4.$$

Мінімальний багаточлен елемента β поля GF(q^m) визначається з вираження:

$$f(x) = \left(x - \beta^{q^0}\right) \cdot \left(x - \beta^{q^1}\right) \cdot \left(x - \beta^{q^2}\right) \ldots \left(x - \beta^{q^{l-1}}\right),$$

де l – найменше ціле число, при якому:

$$\beta^{q^l} = \beta.$$

Значення мінімальних багаточленів будуть наступними:

$$f_1(x) = (x - \alpha^1)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8)(x - \alpha^{16}) =$$
$$= x^5 + x^2 + 1;$$
$$f_2(x) = (x - \alpha^2)(x - \alpha^4)(x - \alpha^8)(x - \alpha^{16})(x - \alpha^1) =$$
$$= x^5 + x^2 + 1;$$
$$f_3(x) = (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^{24})(x - \alpha^{17}) =$$
$$= x^5 + x^4 + x^3 + x^2 + 1;$$
$$f_4(x) = (x - \alpha^4)(x - \alpha^8)(x - \alpha^{16})(x - \alpha^1)(x - \alpha^2) =$$
$$= x^5 + x^2 + 1.$$

Так як $f_1(x) = f_2(x) = f_4(x)$, то:

$$g(x) = f_1(x) \cdot f_3(x) = \left(x^5 + x^2 + 1\right) \cdot \left(x^5 + x^4 + x^3 + x^2 + 1\right) =$$
$$= x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$$

На практиці при визначенні значень багаточлена, що породжує, користуються спеціальною таблицею мінімальних багаточленів (див. таблицю 8 додатка), і вираженням для багаточлена, що породжує:

$$g(x) = f_1(x) f_3(x) \ldots f_j(x).$$

При цьому робота здійснюється в наступній послідовності.

По заданій довжині коду n і кратності помилок, що виправляються, t_u визначають:

– з вираження $n = 2^m - 1$ значення параметра m, що є максимальним ступенем співмножників $g(x)$; – з вираження $j = 2t_u - 1$ максимальний порядок мінімального багаточлена, що входить у число співмножників $g(x)$.

– користуючись таблицею мінімальних багаточленів, визначається вираження для $g(x)$ залежно від m і j. Для цього зі стовпчика, що відповідає параметру m, вибираються багаточлени з порядками від 1 до j, які в результаті перемножування дають значення $g(x)$.

У вираженні для $g(x)$ утримуватися мінімальні багаточлени тільки для непарних ступенів α, тому що звичайно відповідні їм мінімальні багаточлени парних ступенів α мають аналогічні вираження.

Наприклад, мінімальні багаточлени елементів:

$$\alpha^2, \alpha^4, \alpha^8, \ldots$$

відповідають мінімальному багаточлену елемента α^1, мінімальні багаточлени елементів:

$$\alpha^6, \alpha^{12}, \alpha^{24}, \ldots$$

відповідають мінімальному багаточлену α^3 і т.ін.

Необхідно визначити значення багаточлена, що породжує, для побудови примітивного коду над GF(2) довжини 31, що забезпечує $t_u = 3$.

Визначаємо значення m і j.

$$m = \log_2(n+1) = \log_2 32 = 5;$$
$$j = 2t_u - 1 = 2 \cdot 3 - 1 = 5.$$

З таблиці мінімальних багаточленів відповідно до $m = 5$ і $j = 5$ одержуємо :

$$g(x) = 45 \cdot 75 \cdot 67 = (x^5 + x^2 + 1)(x^5 + x^4 + x^3 + x^2 + 1) \cdot$$
$$\cdot (x^5 + x^4 + x^2 + x + 1) = x^{15} + x^{11} + x^{10} + x^9 + x^8 +$$
$$+ x^7 + x^5 + x^3 + x^2 + x + 1.$$

Задані вихідні дані: n і t_u або k і t_u для побудови циклічного коду часто приводять до вибору завищеного значення m і як наслідок цього до невиправданого збільшення довжини коду. Таке положення знижує ефективність отриманого коду, тому що частина інформаційних розрядів взагалі не використовується.

Подібна невідповідність у ряді випадків можна усунути, застосовуючи непримітивний код БЧХ.

Непримітивним кодом БЧХ, що виправляє t_u помилок, називається код довжини n над GF(q), для якого елементи:

$$(\beta^i)^1, (\beta^i)^2, \ldots, (\beta^i)^{2t_u}$$

є коріннями багаточлена, що породжує.

Тут β^i – непримітивний елемент GF(q^m), а довжина коду n дорівнює порядку елемента β^i.

Порядком елемента β^i є найменше n, для якого:

$$(\beta^i)^n = 1.$$

Багаточлен, що породжує, непримітивного коду БЧХ, за аналогією із примітивним кодом, визначається з вираження:

$$g(x) = f_{1i}(x) \cdot f_{3i}(x) \cdots f_{ji}(x),\ f_{1i}(x) \cdot f_{3i}(x) \cdots f_{ji}(x)$$

– мінімальні багаточлени:

$$(\beta^i)^1, (\beta^i)^3, \ldots, (\beta^i)^j$$

елементів поля GF(q^m), які є коріннями $g(x)$; i – ступінь непримітивного елементу β.

Процедура кодування інформації виконується щораз при записі нових даних на диск масштабованої розподіленої файлової системи. Зниження

швидкодії обумовлене необхідністю виконання додаткових операцій над даними. Тому вдосконалення й адаптація алгоритму кодування повинне проводитися убік зменшення часу кодування, пропорційного кількості дискових операцій $N_{кдо}$ (більше значимий критерій) і інструкцій процесора (ключових операцій) $N_{кко}$ (менш значимий).

Процедура виправлення помилок виконується тільки при виникненні помилки і є відносно рідкою подією. Тому, у роботі запропонований як найбільше значимий критерій вибрати реалізовану здатність, що виправляє, алгоритму S, а інші критерії, що мають меншу значимість – кількість дискових операцій $N_{ддо}$ й кількість команд (ключових операцій) $N_{дко}$ при декодуванні, пропонується оптимізувати в останню чергу.

1) У роботі була рекомендована наступна поетапна методика вибору ефективного алгоритму кодування СНЗІ:

– Досліджувати шляхи зменшення кількості дискових операцій при різних варіантах запису кодових груп $N_{кдо}$, скласти список можливих алгоритмів.

– Для кожного варіанта визначити чисельні значення критерію $N_{кдо.}$ Вибрати зі списку алгоритм, що забезпечує мінімальне $N_{кдо}$.

– Досліджувати шляхи зменшення числа ключових операцій, необхідних для обчислення перевірочних розрядів $N_{кко}$, скласти список можливих варіантів реалізацій алгоритмів.

– Визначити чисельні значення критерію $N_{кдо}$ Вибрати варіант (сукупність варіантів) реалізації алгоритму, що дозволяє мінімізувати параметр $N_{кко}$.

2) Рекомендована поетапна методика вибору ефективного алгоритму декодування:

– Досліджувати методи максимізації реалізованої здатності, що виправляє, S, скласти список можливих алгоритмів.

– Визначити чисельні значення критерію S. Вибрати зі списку алгоритм із найкращим значенням S як основу алгоритму декодування двомірного ітеративного коду СНЗІ.

– Досліджувати варіанти реалізації, що дозволяють прискорити процес виправлення помилок по метриках $N_{ддо}$ й $N_{dко}$, скласти список можливих варіантів реалізацій алгоритмів.

– Визначити чисельні значення критерію $N_{ддо}$ й $N_{dко}$ Вибрати варіант (сукупність варіантів) реалізації алгоритму, що дозволяє мінімізувати параметри $N_{ддо}$ й $N_{dко}$.

Для визначення значень кількості дискових операцій використовувався звичайний підрахунок кількості зчитувальних-записуваних секторів, необхідних для виконання операцій запису даних і відновлення інформації.

Критерій S (реалізована здатність, що виправляє) обчислювався відповідно до формул теорії завадостійкого кодування.

Вибір методики визначення чисельних значень метрик критеріїв $N_{кко}$ й $N_{dко}$ проводився відповідно до вимог: ступінь адекватності поставленому завданню одержуваних за допомогою обчислювальної моделі оцінок, трудомісткість подання алгоритмів і незалежність моделі від архітектури й механізмів керування ресурсами операційної системи. Були оцінені наступні обчислювальні моделі й методи оцінки обчислювальної складності:

– Машини Тьюринга.

– Метод зведення завдання.

– Метод оцінки складності з використанням граф-схем алгоритмів.

Показано, що метод побудови граф-схем алгоритмів, є єдиним методом, що дозволяє одержати точні значення критеріїв $N_{кко}$ й $N_{dко}$, і описати дрібні відмінності близьких за структурою алгоритмів, тому в роботі був обраний даний метод.

Розглянемо більш детально кожний з функціональних блоків розробленої програми. Програма записує дані на носій інформації масштабованої розподіленої файлової системи піддаючи їх кодуванню за методом БЧХ. Це робиться для того, щоб при декодуванні, якщо виникла помилка, то цю помилку можливо було локалізувати та виправити.

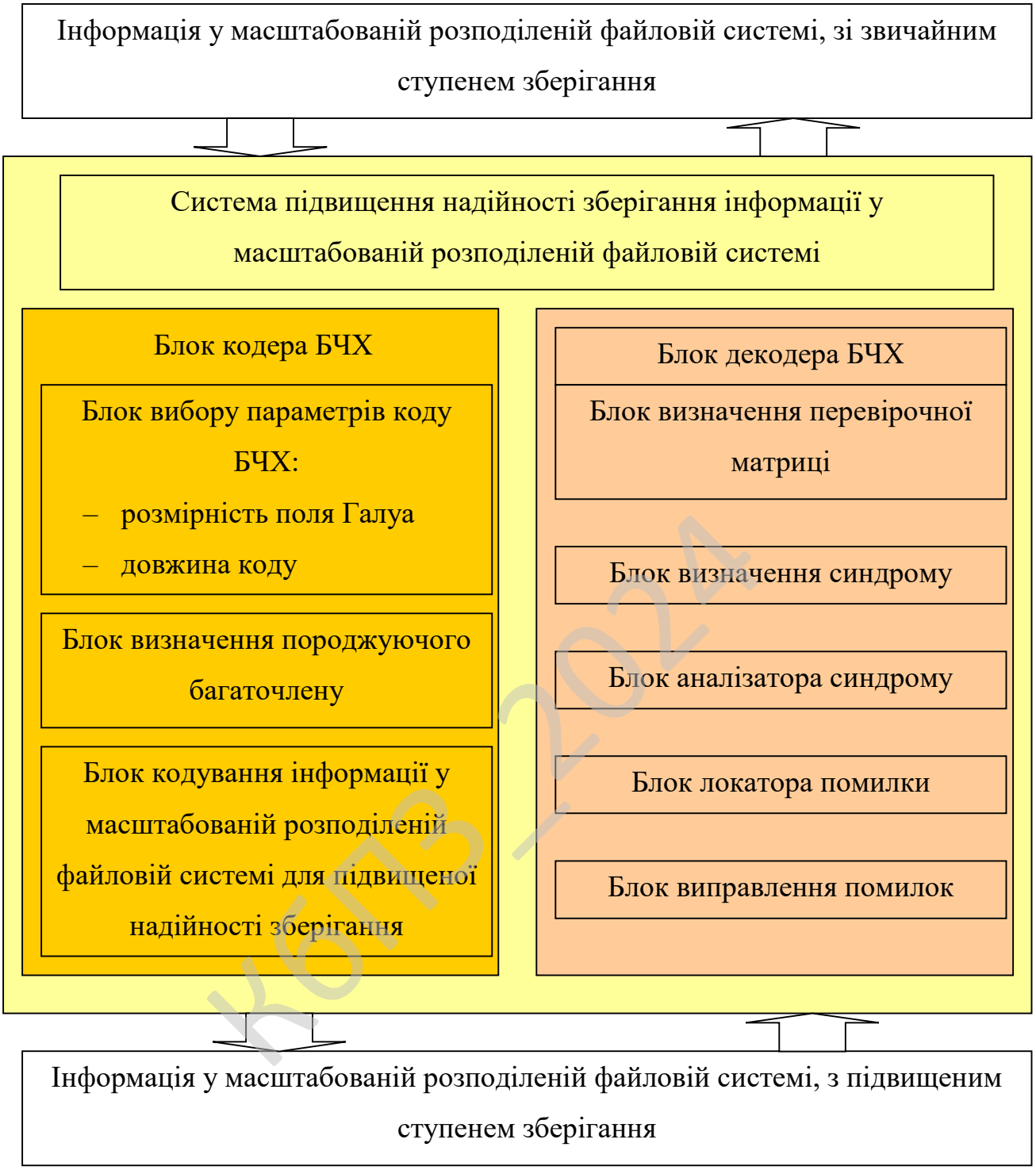

Рисунок 3.2 – Функціональна схема системи

Незакодована інформація подається на блок кодування, де відбуваються наступні дії:

1. Вибираються наступні параметри коду БЧХ, для проведення операції оптимального кодування, згідно з розміром даних:

– розмірність поля Галуа;

– довжина коду;

– мінімальна кодова відстань.

2. Визначається породжуючий багаточлен.

3. Відбувається кодування інформації для підвищеної надійності зберігання

Після виконання цих дій, інформація зберігається у закодованому вигляді, що дає змогу підвищити рівень її надійності зберігання.

Для того, що прочитати інформацію вона поступає на декодер кодів БЧХ, який виконує наступні дії:

1. Згідно параметрів кодування відбувається визначення перевірочної матриці.

2. Розраховується синдром помилок.

3. Проводиться аналізатор синдрому.

4. Якщо є помилка то вона локалізується.

5. Якщо помилки є то вони виправляються.

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

3.4 Розробка діаграми процесів

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.

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі.

Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

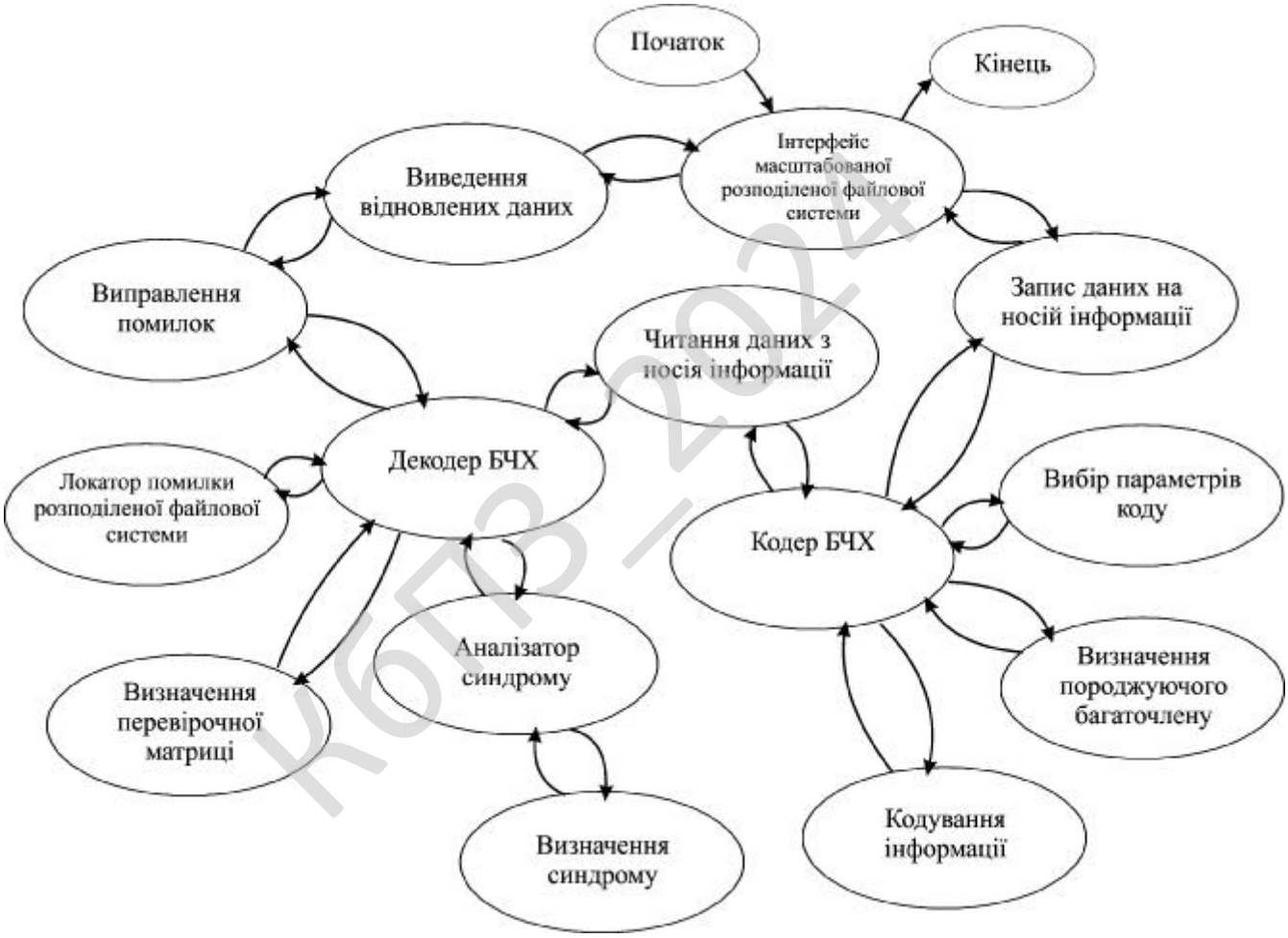

Рисунок 3.3 – Діаграма взаємодії процесів

Діаграми потоків даних містять чотири типи елементів:

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

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схеми є основою ПЗ. Тому від точності і детальності проробки блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації, також те, що при розробці програми слід надати особливу увагу модулю масштабованої розподіленої файлової системи.

Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

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

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 – перевірки поточного стану та поверненням на початок схеми чи з завершенням роботи розробленого ПЗ.

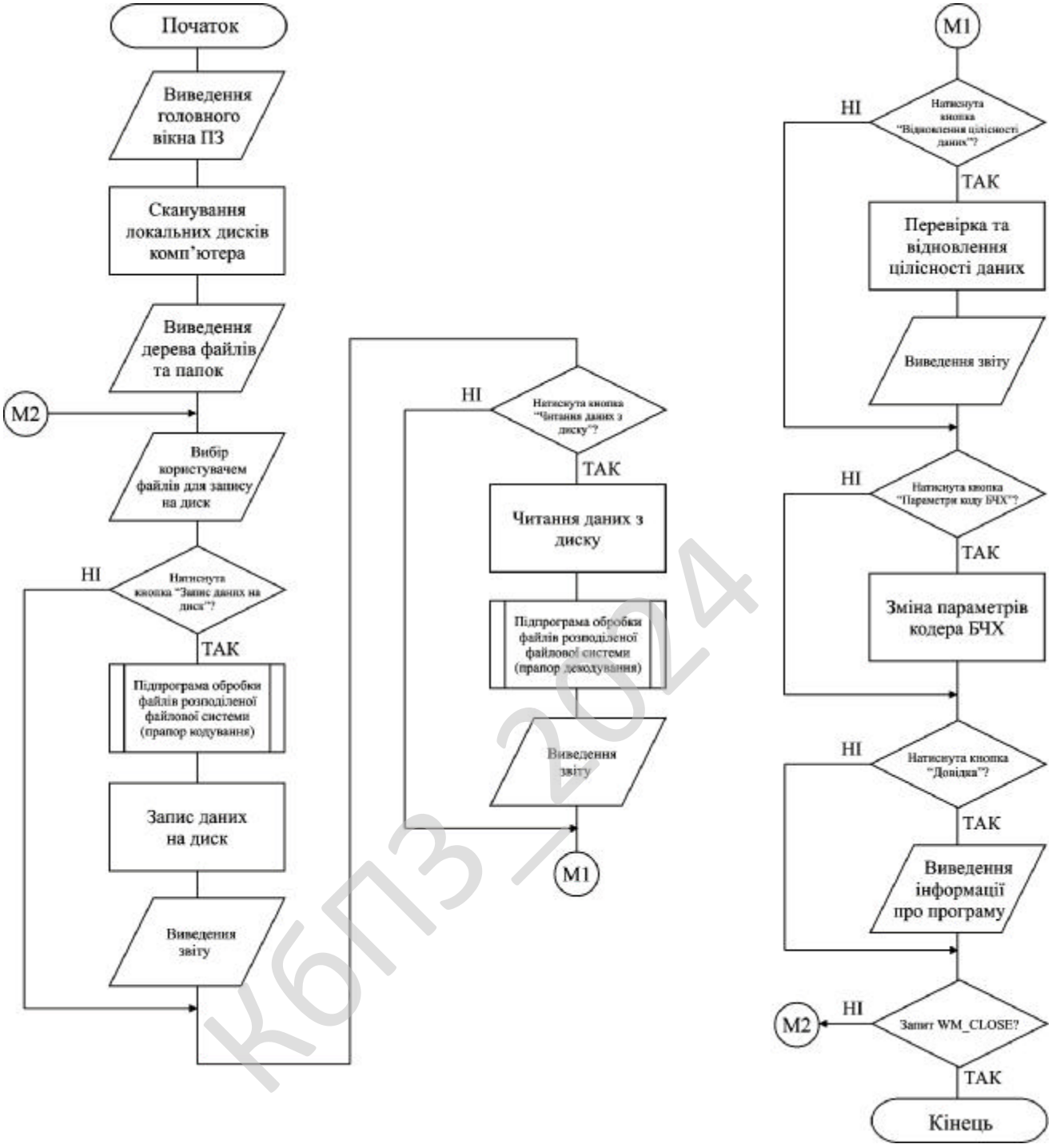

Рисунок 4.1 – Блок-схема основної програми

При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

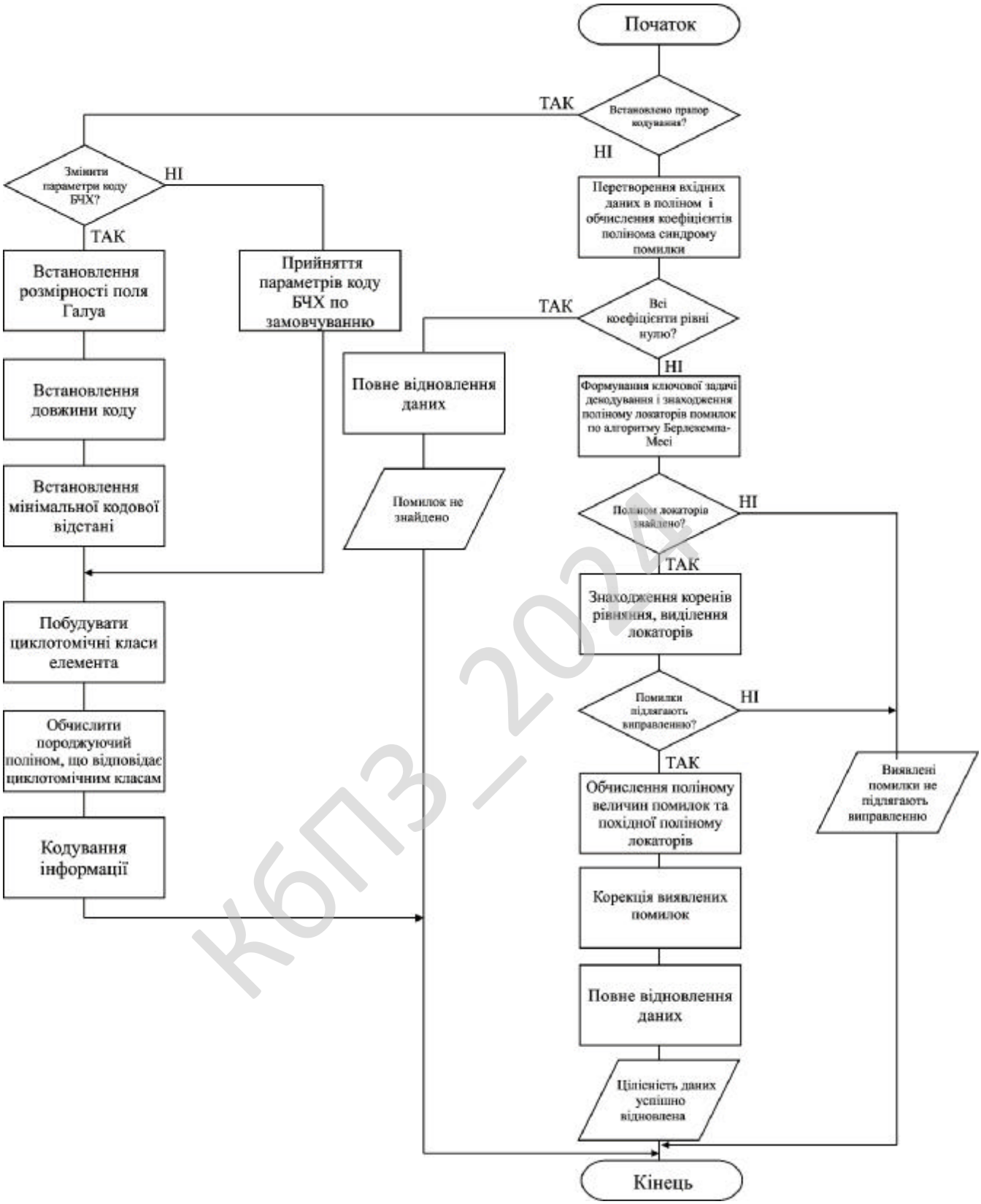

Рисунок 4.2 – Блок-схема роботи підпрограми

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.

Розглянемо використані технології та їх основні компоненти що підтверджують правильність використаних проектних рішень.

Незважаючи на те що я працював над ПЗ один в реалізації програми я використовував підходи пришвидшення розробки на основі методологій Agile.

Гнучка́ розробка програмного забезпечення (Agile software development, agile-методи) – клас методологій розробки програмного забезпечення, що базується на ітеративній розробці, в якій вимоги та розв'язки еволюціонують через співпрацю між самоорганізовуваними багатофункціональними командами.

Гнучка розробка – найкращий засіб для підвищення продуктивності розробників програмного забезпечення.

Більшість гнучких методологій націлені на мінімізацію ризиків, шляхом зведення розробки до серії коротких циклів, що мають назву ітерацій, які зазвичай тривають один-два тижні. Кожна ітерація сама по собі виглядає як програмний проект в мініатюрі, і включає всі завдання, необхідні для видачі мінімального приросту за функціональністю: планування, аналіз вимог, проектування, кодування, тестування і документування. Хоча окрема ітерація, як правило, недостатня для випуску нової версії продукту, мається на увазі те, що гнучкий програмний проект готовий до випуску наприкінці кожної ітерації. Після закінчення кожної ітерації, команда виконує переоцінку пріоритетів розробки.

Agile акцентує увагу на безпосередньому спілкуванні «віч-на-віч». Більшість agile команд розташовані в одному офісі, його іноді називають bullpen. Як мінімум вона включає і «замовників» (замовники, які визначають продукт, також це можуть бути менеджери продукту, бізнес аналітики або клієнти). Офіс може також включати тестувальників, дизайнерів інтерфейсу, технічних авторів і менеджерів.

Основною метрикою agile методів є робочий продукт. Віддаючи перевагу безпосередньому спілкуванню, agile-методи зменшують обсяг письмової документації в порівнянні з іншими методами. Це привело до критики цих методів як недисциплінованих.

Agile – родина процесів розробки, а не єдиний підхід в розробці програмного забезпечення, і визначається Agile Manifesto. Agile не включає практик, а визначає цінності та принципи, якими керуються успішні команди.

Agile Manifesto розроблений і прийнятий 17 розробниками 11-13 лютого 2001 року на лижному курорті The Lodge at Snowbird в горах Юти. Маніфест підписали представники наступних методологій Extreme programming, Scrum, DSDM, Adaptive software development, Crystal Clear, Feature driven development, Pragmatic Programming. Agile Manifesto містить 4 основні ідеї та 12 принципів. Примітно, що Agile Manifesto не містить практичних порад.

Основні ідеї:

– Особистості та їхні взаємодії важливіші, ніж процеси та інструменти.

– Робоче програмне забезпечення важливіше, ніж повна документація.

– Співпраця із замовником важливіша, ніж контрактні зобов'язання.

– Реакція на зміни важливіша, ніж дотримання плану.

Принципи, які роз'яснює Agile Manifesto:

– задоволення клієнта за рахунок ранньої та безперебійної поставки коштовного програмного забезпечення;

– вітання змін вимог навіть наприкінці розробки (це може підвищити конкурентоспроможність отриманого продукту);

– часта поставка робочого програмного забезпечення (кожен місяць або тиждень або ще частіше);

– тісне, щоденне спілкування замовника з розробниками впродовж всього проекту;

– проектом займаються мотивовані особистості, які забезпечені потрібними умовами роботи, підтримкою і довірою;

– рекомендований метод передачі інформації – особиста розмова (віч-на-віч);

– робоче програмне забезпечення – найкращий вимірювач прогресу;

– спонсори, розробники та користувачі повинні мати можливість підтримувати постійний темп на невизначений термін;

– постійну увагу поліпшенню технічної майстерності та зручному дизайну;

– простота – мистецтво не робити зайвої роботи;

– найкращі технічні вимоги, дизайн та архітектура виходять у самоорганізованої команди;

– постійна адаптація до мінливих обставин.

Маніфест та Принципи гнучкої розробки містять високорівневі ідеї щодо того, як потрібно вибудовувати процес розробки програмного забезпечення, щоб успішно завершувати проекти й створювати команди, в яких приємно та цікаво працювати.

Документи визначають, що потрібно для цього зробити, але не говорять, як це зробити. По-іншому й не могло бути, оскільки Маніфест та Принципи народилися внаслідок консенсусу представників різних (хоча й споріднених) напрямів, які могли знайти спільну основу лише на рівні базових цінностей та принципів.

Критика. Багато керівників проектів, що працюють у традиційних методологіях на кшталт «водоспаду», критикують agile-методи.

Один з повторюваних пунктів критики: при agile-підході часто нехтують створенням «дорожньої карти» розвитку продукту, так само як і управлінням

вимогами, в процесі якого і формується така «карта». Гнучкий підхід до управління вимогами не має на увазі далекосяжних планів (по суті, управління вимогами просто не існує в даній методології), а має на увазі можливість замовника раптом і несподівано наприкінці кожної ітерації виставляти нові вимоги, що часто суперечать архітектурі вже створеного і поставленого продукту. Таке іноді призводить до катастрофічних «авралів» з масовим рефакторингом і переробками практично на кожній черговій ітерації.

Крім того вважається, що робота в agile мотивує розробників вирішувати всі прибулі завдання найпростішим і найшвидшим можливим способом, при цьому часто не звертаючи уваги на коректність коду з точки зору вимог базової платформи (підхід «працює, та й добре», при цьому не враховується, що може перестати працювати при найменшій зміні або ж породити важкі до відтворення дефекти після реального розгортання у клієнта). Це призводить до зниження якості продукту і накопиченню дефектів.

Методології. Існують методології, які дотримуються цінностей і принципів заявлених в Agile Manifesto, деякі з них:

1. Agile Modeling – набір понять, принципів і прийомів (практик), що дозволяють швидко і просто виконувати моделювання і документування в проектах розробки програмного забезпечення. Не включає в себе детальну інструкцію з проектування, не містить описів, як будувати діаграми на UML.

Основна мета – ефективне моделювання і документування; але не охоплює програмування та тестування, не включає питання управління проектом, розгортання і супроводу системи. Однак включає в себе перевірку моделі кодом.

2. Agile Unified Process (AUP) спрощена версія IBM Rational Unified Process (RUP), розроблена Скоттом Амблером, яка описує просте і зрозуміле наближення (модель) для створення програмного забезпечення для бізнес-додатків.

3 Agile Data Method – група ітеративних методів розробки програмного забезпечення, в яких вимоги та рішення досягаються в рамках співпраці різних крос-функціональних команд.

4. DSDM заснований на концепції швидкої розробки додатків (Rapid Application Development, RAD). Являє собою ітеративний і інкрементний підхід, який надає особливого значення тривалій участі в процесі користувача/споживача.

5. Essential Unified Process (EssUP).

6. Екстремальне програмування (Extreme programming, XP).

7. Feature driven development (FDD) – функціонально-орієнтована розробка. Використовуване в FDD поняття функції або властивості (feature) системи, досить близько до поняття прецеденту використання, використовуваному в RUP, істотна відмінність – це додаткове обмеження: «кожна функція повинна допускати реалізацію не більше, ніж за два тижні». Тобто якщо сценарій використання досить малий, його можна вважати функцією. Якщо ж великий, то його треба розбити на декілька відносно незалежних функцій.

8. Getting Real – ітераційний підхід без функціональних специфікацій, що використовується для веб-додатків. У даному методі спершу розробляється інтерфейс програми, а потім її функціональна частина.

9. OpenUP – це ітераційно-інкрементний метод розробки програмного забезпечення. Позиціюється, як легкий і гнучкий варіант RUP. OpenUP ділить життєвий цикл проекту на чотири фази: початкова фаза, фази уточнення, конструювання та передачі. Життєвий цикл проекту забезпечує надання зацікавленим особам та членам колективу точок ознайомлення і прийняття рішень впродовж усього проекту. Це дозволяє ефективно контролювати ситуацію і вчасно приймати рішення про задовільність результатів. План проекту визначає життєвий цикл, а кінцевим результатом є остаточний додаток.

10. Scrum встановлює правила керування процесом розробки та дозволяє використовувати вже існуючі практики кодування, коректуючи вимоги або вносячи тактичні зміни. Використання цієї методології дає можливість виявляти і усувати відхилення від бажаного результату на більш ранніх етапах розробки програмного продукту.

11. Бережлива розробка програмного забезпечення (lean software development). Використовує підходи з концепції бережливого виробництва.

Jira – була використана комерційна система відслідковування помилок, призначена для організації взаємодії з користувачами, хоча в деяких випадках використовується і для управління проектами. Розроблено компанією Atlassian, є одним з двох її основних продуктів (поряд з вікі-системою Confluence). Має веб-інтерфейс.

Назва системи отримано шляхом усічення слова «Gojira» – Японського імені монстра Годзилла, що, в свою чергу, є відсиланням до назви конкуруючого продукту – Bugzilla; створювалася в якості заміни Bugzilla і багато в чому повторює її архітектуру. Система дозволяє працювати з декількома проектами. Для кожного з проектів створює і веде схеми безпеки і схеми оповіщення.

До версії 3.13.5 (включно) розрізнялися редакції Enterprise, Professional і Standard, після – Залишилася тільки редакція Enterprise (для великих організацій).

Система заснована на Java EE і працює на кількох популярних системах управління базами даних і операційних системах.

Основний елемент обліку в системі – завдання (ticket або issue). Завдання містить назву проекту, тему, тип, пріоритет, компоненти і зміст. Завдання може бути розширена додатковими полями (також і нові призначені для користувача поля можуть бути визначені), додатками (наприклад – Фотографіями, скриншотами) або коментарями. Завдання може редагуватися або просто змінювати статус, наприклад, з «відкритий» в «закритий». Які переходи між станами можливі, визначається через настроюється потік операцій. Будь-які зміни в задачі записуються в журнал.

Jira має велику кількість можливостей конфігурації: для кожної програми може бути визначений окремий тип завдання з власним workflow, набором статусів, одним або декількома видами уявлення (screens). Крім того, за допомогою так званих «схем» можна визначити для кожного індивідуального Jira-проекту власні права доступу, поведінку і видимість полів і багато іншого.

Завдяки універсальному підходу можна пристосувати Jira для багатьох непрофільних завдань, наприклад, керування вимогами, керування ризиками, аж до реалізації невеликої системи бронювання, автоматизації процесу рекрутингу.

Для інтеграції з зовнішніми системами підтримує інтерфейси SOAP, XML-RPC і REST. Поставляється із засобами інтеграції з такими системами управління версіями, як Subversion, CVS, Git, Clearcase, Team Foundation Server, Mercurial і Perforce.

Існують доповнення, що дозволяють вбудувати Jira в інтегровані середовища розробки, в тому числі Eclipse і IntelliJ IDEA. Перекладена багатьма мовами, включаючи російську, англійську, японську, німецьку, французьку, іспанську.

Для сторонніх розробників надаються кошти розробки розширень системи – плагінів. Розробники розширень можуть викладати плагіни для продажу на спеціальний розділ сайту Atlassian.

Є комерційним продуктом, який може бути ліцензований для роботи на локальному сервері або доступний в якості віддаленого додатки. Ціноутворення залежить від максимального числа користувачів, при цьому близько $50 за користувача для локального і $7 на місяць за користувача для віддаленого доступу є типовими цінами.

Для академічних і комерційних клієнтів доступний повний вихідний код під ліцензією розробника.

Для проектів з відкритим вихідним кодом Atlassian надає спеціальну безкоштовну ліцензію при дотриманні наступних правил:

– проект використовує ліцензії, схвалені Open Source Initiative;

– вихідний код проекту доступний для скачування;

– у проекту є публічно доступна веб-сайт;

– програмне забезпечення від Atlassian є на веб-сайті проекту.

Розглянемо побудову циклотомічних класів елемента. Наступним етапом є обчислення породжую чого полінома, що відповідає циклотомічним класам.

Для знаходження полінома, що породжує, необхідно виконати кілька етапів:

1. Визначити параметри коду.

– вибрати q, тобто поле $GF(q)$, над яким буде побудований код;

– вибрати довжину n коду з умови $n = (q^m - 1) / s$, де m, s – цілі позитивні числа;

– задати величину d конструктивної відстані.

2. Побудувати циклотомічні класи елемента $\beta = \alpha^s$ поля $GF(q^m)$ над полем $GF(q)$, де α – примітивний елемент $GF(q^m)$.

3. Оскільки кожному такому циклотомічному класу відповідає полином, що не приводиться, над $GF(q)$, корінням якого є елементи цього й тільки цього класу, зі ступенем рівним кількості елементів у класі, то вибрати:

$$\beta^{l_0}, \beta^{l_0+1}, \ldots, \beta^{l_0+d-2},$$

таким чином, щоб сумарна довжина циклотомічних класів була мінімальна.

4. Обчислити поліном, що породжує:

$$g(x) = f_1(x) f_2(x) \ldots f_h(x),$$

де $f_i(x)$ – поліном, що відповідає i-ому циклотомичному класу.

Після обчислення поліному, відбувається кодування інформації.

Розглянувши операцію кодування, перейдемо до розгляду операції декодування кодами БЧХ.

З нього ми бачимо, що підпрограма реалізується послідовністю виконання наступних ітерацій.

Спершу відбувається перетворення вхідних даних у полінома обчислення коефіцієнтів полінома синдрому помилки.

Після цього відбувається перевірка рівності усіх коефіцієнтів нулю.

Якщо вони усі дорівнюють нулю то відбувається наступні дії:

– Відбувається повне відновлення даних.

– Виводиться повідомлення про те, що помилок не знайдено.

– Підпрограма завершує свою роботу.

У іншому випадку. Тобто, якщо не усі коефіцієнти дорівнюють нулю, то відбувається наступна послідовність дій.

Спершу формується ключова задача декодування та знаходження поліному локаторів помилок по алгоритму Берклемпа-Мессі.

Якщо поліном локаторів не знайдено, то виводиться повідомлення, про те, що виявлені помилки не підлягають виправленню.

Якщо ж поліном локаторів знайдено, то відбувається знаходження коренів рівняння, та виділяються локатори.

Після цього відбувається перевірка на виправленість помилок.

Якщо помилки неможливо виправити, то виводиться повідомлення, про те, що виявлені помилки не підлягають виправленню.

Якщо ж помилки можливо виправити, то відбувається послідовність наступних кроків:

– Обчислюється поліном величини помилок та похідна поліному локаторів.

– Відбувається корекція виявлених помилок.

– Відбувається повне відновлення даних.

– Виводиться повідомлення про те, що цілісність даних успішно відновлена.

Алгоритм Берклемпа-Мессі, працює наступним чином.

Задати необхідну послідовність біт:

$$s_0, s_1, ..., s_{n-1}.$$

Створити масиви b, t, cдовжини n, задати початкові значення:

$$b_0 \leftarrow 1, c_0 \leftarrow 1, N \leftarrow 1, L \leftarrow 0, m \leftarrow -1.$$

Поки $N < n$:

1. Обчислити:

$$d \leftarrow s_N \oplus c_1 s_{N-1} \oplus c_2 s_{N-2} \oplus ... \oplus c_L s_{N-L}.$$

2. Якщо $d = 0$, то поточна функція генерує обрану ділянку:

$$s_{N-L}, s_{N-L+1}, \ldots, s_N$$

послідовності; залишити функцію колишньої.

3. Якщо $d \neq 0$:

– Зберегти копію масиву cв.t

– Обчислити нові значення:

$$c_{N-m} \leftarrow c_{N-m} \oplus b_0, c_{N-m+1} \leftarrow c_{N-m+1} \oplus b_1, \ldots, c_{n-1} \leftarrow c_{n-1} \oplus b_{n-N+m-1}$$

.

– Якщо $2L \leqslant N$, установити значення:

$$L \leftarrow N + 1 - L, m \leftarrow N,$$

і скопіювати t в b.

4. $N \leftarrow N + 1$.

У результаті масив c – функція зворотного зв'язка, тобто:

$$c_L s_i \oplus c_{L-1} s_{i+1} \oplus c_{L-2} s_{i+2} \oplus \ldots \oplus c_0 s_{i+L} = 0,$$

для будь-яких i.

Реалізація коду на Visual C#:

```
byte[] b, c, t, s;
int N, L, m;
 public void BerlekampRegistryTester(int sequenceLength)
{
    b = new byte[sequenceLength];
    c = new byte[sequenceLength];
    t = new byte[sequenceLength];
    s = new byte[sequenceLength];
    for (int i = 0; i < sequenceLength; i++)
        b[i] = c[i] = t[i] = 0;
    b[0] = c[0] = 1;
    N = L = 0;
    m = -1;
}
private void runTest()
{
    int d;
    while (N < s.Length)
    {
        d = 0;
        for (int i = 0; i <= L; i++)
```

```
                d += s[ N-i] * c[i];
            d = d % 2;
            if (d != 0)
            {
                t = (byte[])c.Clone();
                for (int i = 0; i <= s.Length + m - 1 - N; i++)
                    c[N - m + i] = (byte)(c[N - m + i] ^ b[i]);
                if (L <= (N / 2))
                {
                    L = N + 1 - L;
                    m = N;
                    b = (byte[])t.Clone();
                }
            }
            N++;
        }
    }
```

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм RC5, який являє собою блоковий шифр із безліччю параметрів: розміром блоку, розміром ключа й числом раундів. В алгоритмі RC5 передбачені три операції: XOR, додавання й циклічні зрушення. На більшості процесорів операції циклічного зрушення виконуються за постійний час, змінні циклічні зрушення являють собою нелінійну функцію. Циклічні зрушення залежать як від ключа, так і від даних.

В RC5 використовується блок змінної довжини, але в приводиться прикладі, що буде розглянутий, 64-бітовий блок даних. Шифрування використовує $2r+2$ залежних від ключа 32-бітових слів – S_0, S_1, S_2,... S_{2r+1} – де r – число раундів. Для шифрування спочатку потрібно розділити блок відкритого тексту на два 32-бітових слова: A и B. (При впакуванні байтів у слова в алгоритмі RC5 дотримується угода про прямий порядок (little-endian) байтів: перший байт займає молодші біти регістра A й т. ін.) Потім:

$$A=A + S_0$$

$$B = B + S_0$$

Для і від 1 до r:

$$A = ((A \oplus \quad B) <<< B) + S_{2i}$$

$$B = ((B \oplus \quad A) <<< A) + S_{2i+1}$$

Вихід перебуває в регістрах A и B.

Розшифрування теж нескладно. Потрібно розбити блок відкритого тексту на два слова, A й B, а потім:

Для i від r до 1 із кроком -1:

$$B = ((B - S_{2i+1}) >>> A) \oplus A$$

$$A = ((A - S_{2i}) >>> B) \oplus B$$

$$B = B - S_i$$

$$A = A - S_0$$

Символом «>>>» позначене циклічне зрушення вправо**.** Звичайно ж**,** всі додавання й вирахування виконуються по модулю 2^{32}.

Створення масиву ключів складніше, але теж прямолінійно. Спочатку байти ключа копіюються в масив L із 32-бітових слів, доповнюючи при необхідності заключне слово нулями. Потім масив S ініціалізується за допомогою лінійного конгруентного генератора по модулю 2^{32}:

$$S_0 = P$$

Для i від 1 до $2(r + 1) - 1$:

$$S_i = (S_{i-1} + Q) \bmod 2^{32}$$

де P = 0xb7e15163 і Q = 0x9e3779b9.

Нарешті, потрібно підставити L в S:

$$i = j = 0$$

$$A = B = 0$$

Виконати $3n$ раз (де n – максимум від $2(r + 1)$ і c):

$$A = S_i = (S_i + A + B) <<< 3$$

$$B = L_i = (L_i + A + B) <<< (A + B)$$

$$i = (i + 1) \bmod 2(r + 1)$$

$$i = (j + 1) \bmod c$$

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ масштабованої розподіленої файлової системи яке зображено на рисунку 5.1. Розрахунок проводиться через консоль, а результати передаються до графічного інтерфейсу. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційне меню: Файл; Студент; Опції; 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.

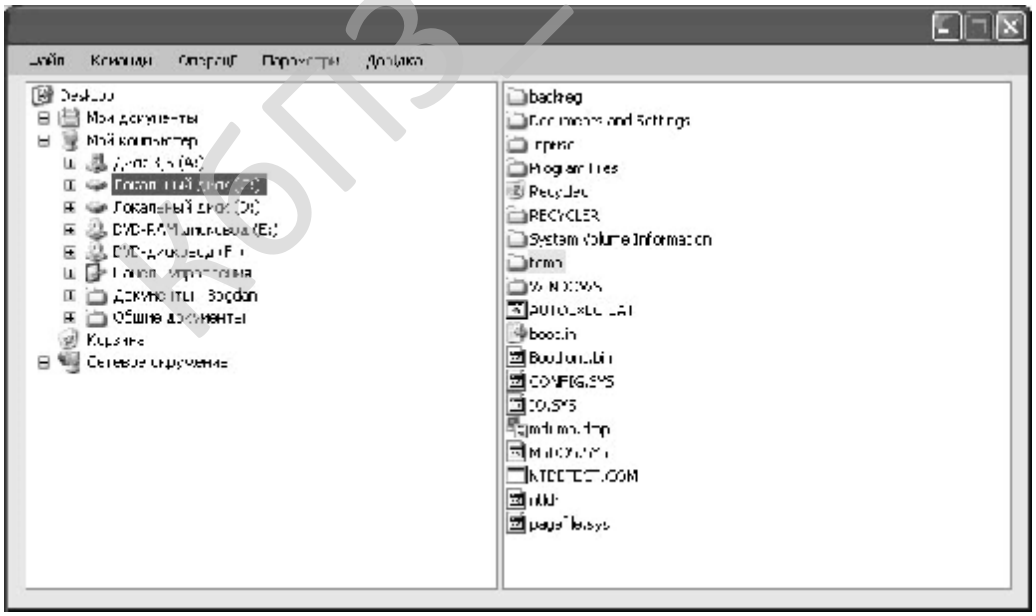

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і інтуїтивно зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий.

Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

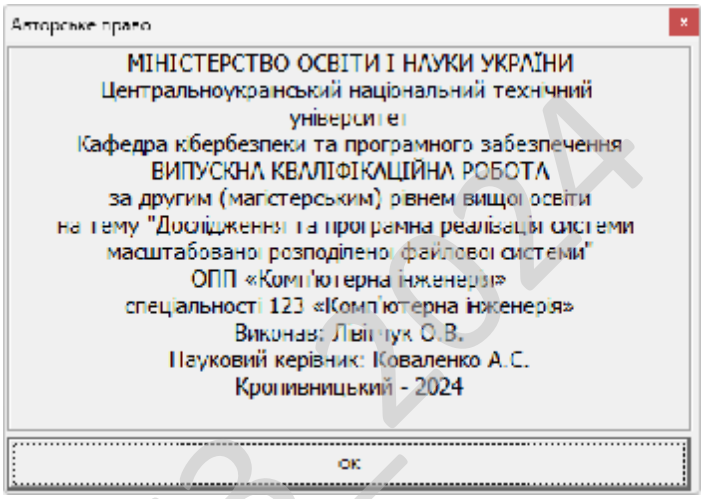

Рисунок 5.2 – Авторське право

Розглянемо процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є

унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

– Виділення критичних, з точки зору загального результату, процедур в діяльності організації. Коли набір таких процедур визначений, необхідно в першу чергу використовувати IT рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене IT рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження;

– Розширення нормативної бази організації шляхом включення до неї регламентів, що описують порядок виконання процедур автоматизованих процесів. В іншому випадку є небезпека виникнення неузгодженості між автоматизованими процедурами та іншими процесами організації.

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються

в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

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

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

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

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

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

Проводилось тестування чорної скриньки. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

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

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

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

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

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

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

Будь-який спосіб тестування «чорної скриньки» повинен:

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

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

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

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій;

– Помилок інтерфейсу;

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

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

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Shareware.

Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму.

В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи масштабованої розподіленої файлової системи.

Метою розробки є дослідження та програмна реалізація системи масштабованої розподіленої файлової системи.

Об'єктом дослідження є процес масштабованої розподіленої файлової системи.

Предметом дослідження є методи масштабованої розподіленої файлової системи.

Методи дослідження базуються на методах файлових систем, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод масштабованої розподіленої файлової системи.

– Розроблено вітчизняний продукт масштабованої розподіленої файлової системи, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та програмної реалізації системи масштабованої розподіленої файлової системи можуть бути цікавими для досить широкої цільової аудиторії (рисунок 7.1).

1. корпорації та підприємства з великими обсягами даних	**хмарні провайдери**: компанії, що надають хмарні послуги (наприклад, aws, google cloud, microsoft azure), зацікавлені в розробці ефективних і масштабованих розподілених файлових систем для оптимізації зберігання та управління даними.
	фінансові установи: банки та фінансові компанії часто оперують великими обсягами конфіденційної інформації та потребують системи з високим рівнем надійності, безпеки та швидкого доступу до даних.
	медіакомпанії: зберігання та управління великими обсягами мультимедійних файлів вимагає масштабованих розподілених систем для забезпечення швидкої доставки контенту.
2. інститути та дослідницькі центри	**дослідницькі установи з великими обчислювальними потребами**: університети та лабораторії, що працюють із високопродуктивними обчисленнями (hpc), для обробки наукових даних (генетичні дослідження, фізика, космос) потребують системи, здатної зберігати та швидко обробляти великі обсяги даних.
	центри обробки великих даних: інститути, що спеціалізуються на big data та штучному інтелекті, мають постійний попит на масштабовані розподілені файлові системи для підтримки швидкого доступу до великих обсягів даних.
3. компанії у сфері кібербезпеки	**постачальники рішень з кібербезпеки**: високоефективні розподілені файлові системи, які забезпечують захист даних та їх резервне копіювання, мають важливе значення для компаній, що забезпечують безпеку корпоративних і персональних даних.
	державні установи: організації, що займаються національною безпекою та потребують управління великими обсягами конфіденційних даних, також зацікавлені у впровадженні розподілених файлових систем.
4. ІТ-компанії та стартапи	**розробники пз для хмарних і децентралізованих систем**: стартапи, що працюють над технологіями розподілених систем або децентралізованих мереж (наприклад, blockchain), можуть використовувати результати для покращення продуктивності та ефективності своїх продуктів.
	виробники програмного забезпечення для IoT в умовах швидкого зростання інтернету речей (iot), де важливим є обмін даними між пристроями, розподілена файлова система забезпечить зберігання та управління даними на високому рівні.
5. державні установи та освітні заклади	**державні організації**: впровадження розподілених файлових систем може допомогти державним організаціям зберігати великі масиви даних, підвищуючи ефективність роботи та доступність інформації.
	освітні установи: університети, що володіють значними базами освітніх ресурсів, можуть використовувати розподілені файлові системи для забезпечення доступу до навчальних матеріалів та ресурсів.
6. платформи соціальних мереж та онлайн-контенту	**соціальні мережі**: компанії, що керують соціальними мережами (наприклад, meta, twitter), щоденно обробляють величезні обсяги зображень, відео та текстових даних, для чого потрібні швидкодоступні, масштабовані та надійні файлові системи.
	платформи потокового мовлення: компанії, що займаються потоковим відео та музикою (наприклад, netflix, spotify), також потребують високоефективних розподілених систем для управління мультимедійними файлами.

Рисунок 7.1 – Цільова аудиторія

Впровадження таких систем дозволяє цим організаціям ефективніше використовувати ресурси, знижувати витрати на зберігання, покращувати безпеку та забезпечувати швидкий доступ до даних у великих масштабах.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для оцінки привабливості проекту програмної реалізації системи масштабованої розподіленої файлової системи можна використати метод експертних оцінок. При цьому експерти оцінюють проект за обраними важливими критеріями (рисунок 7.2).

Рисунок 7.2 – Критерії оцінювання

Експерти виставляють бали за кожним критерієм за шкалою від 1 до 10, де 10 балів означають високу привабливість. Результати зводимо в таблицю 7.1.

Загальний середній бал для проекту можна визначити як середнє арифметичне значення оцінок усіх критеріїв:

Загальний середній бал=8.6+9.4+8.0+7.0+9.0+6.46=8.07

Загальний середній бал проекту становить 8.07, що вказує на високу привабливість проекту. Найвищі оцінки отримали критерії «Масштабованість та

гнучкість» та «Безпека даних», що відображає важливі конкурентні переваги проекту. Натомість вартість впровадження отримала порівняно низький бал, що може вказувати на необхідність додаткового аналізу витрат або пошук можливостей для їх оптимізації.

Таблиця 7.1 – Зведені результати експертних оцінок

Критерій	Експерт 1	Експерт 2	Експерт 3	Експерт 4	Експерт 5	Середня оцінка
Технологічна надійність	8	9	9	8	9	8.6
Масштабованість та гнучкість	9	10	9	10	9	9.4
Ринкова потреба	8	8	9	7	8	8.0
Економічна ефективність	7	6	8	7	7	7.0
Безпека даних	9	9	10	9	8	9.0
Вартість впровадження	6	7	6	7	6	6.4

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості програмної реалізації системи масштабованої розподіленої файлової системи доцільно застосувати методи, які враховують складність проекту, витрати на ресурси та перспективи його розвитку. Найбільш доцільним для оцінки вартості системи масштабованої розподіленої файлової системи може бути COCOMO II або Life Cycle Costing (рис. 7.3).

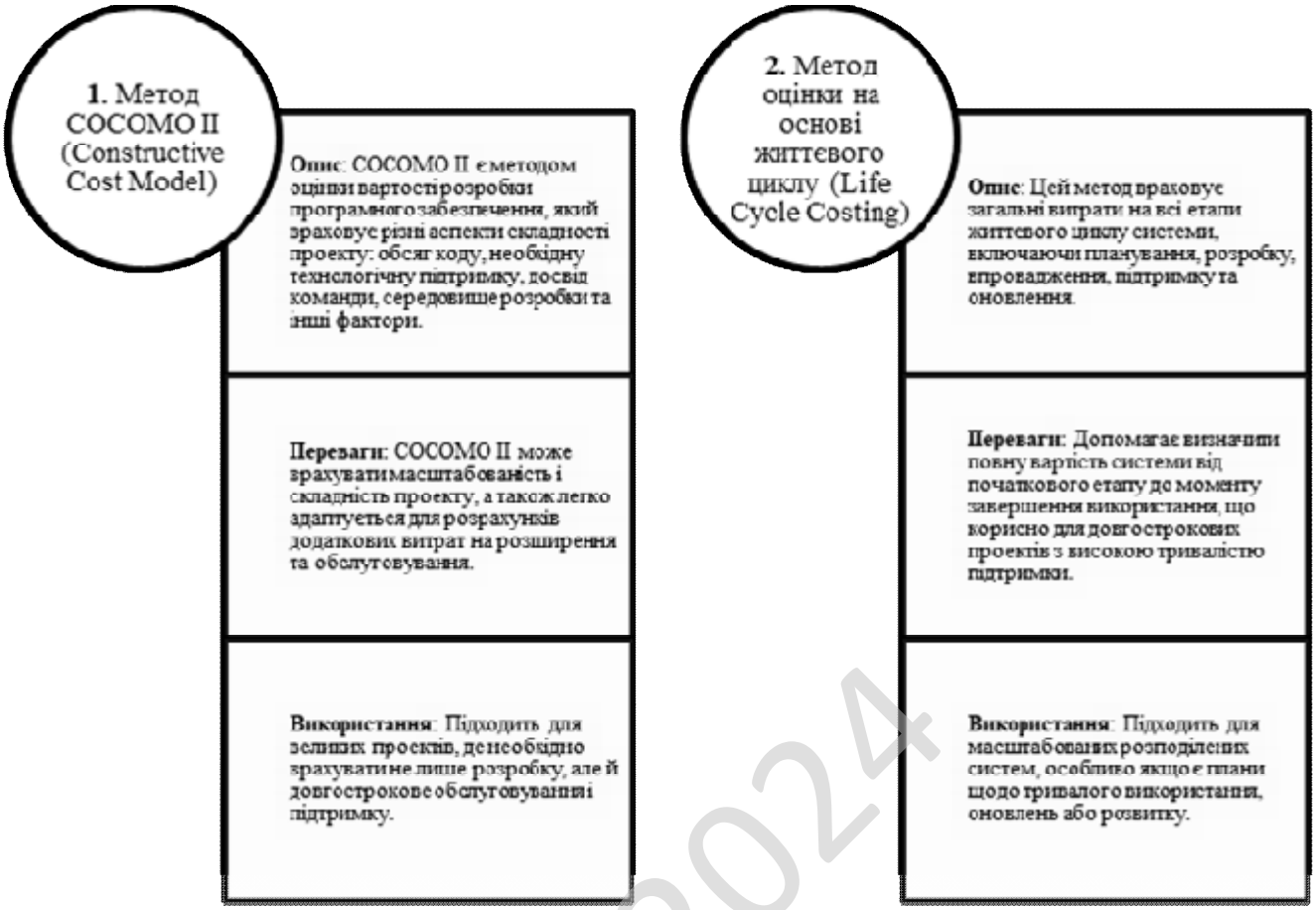

Рисунок 7.3 – Методи оцінювання проєкту

COCOMO II забезпечить більш детальний аналіз витрат на розробку з урахуванням складності, а Life Cycle Costing дозволить розрахувати повні витрати на розробку, впровадження та обслуговування системи.

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

Впровадження масштабованої розподіленої файлової системи може мати значний економічний ефект, особливо для компаній, що обробляють великі обсяги даних. Ось приклад можливих показників економічної ефективності від такого впровадження:

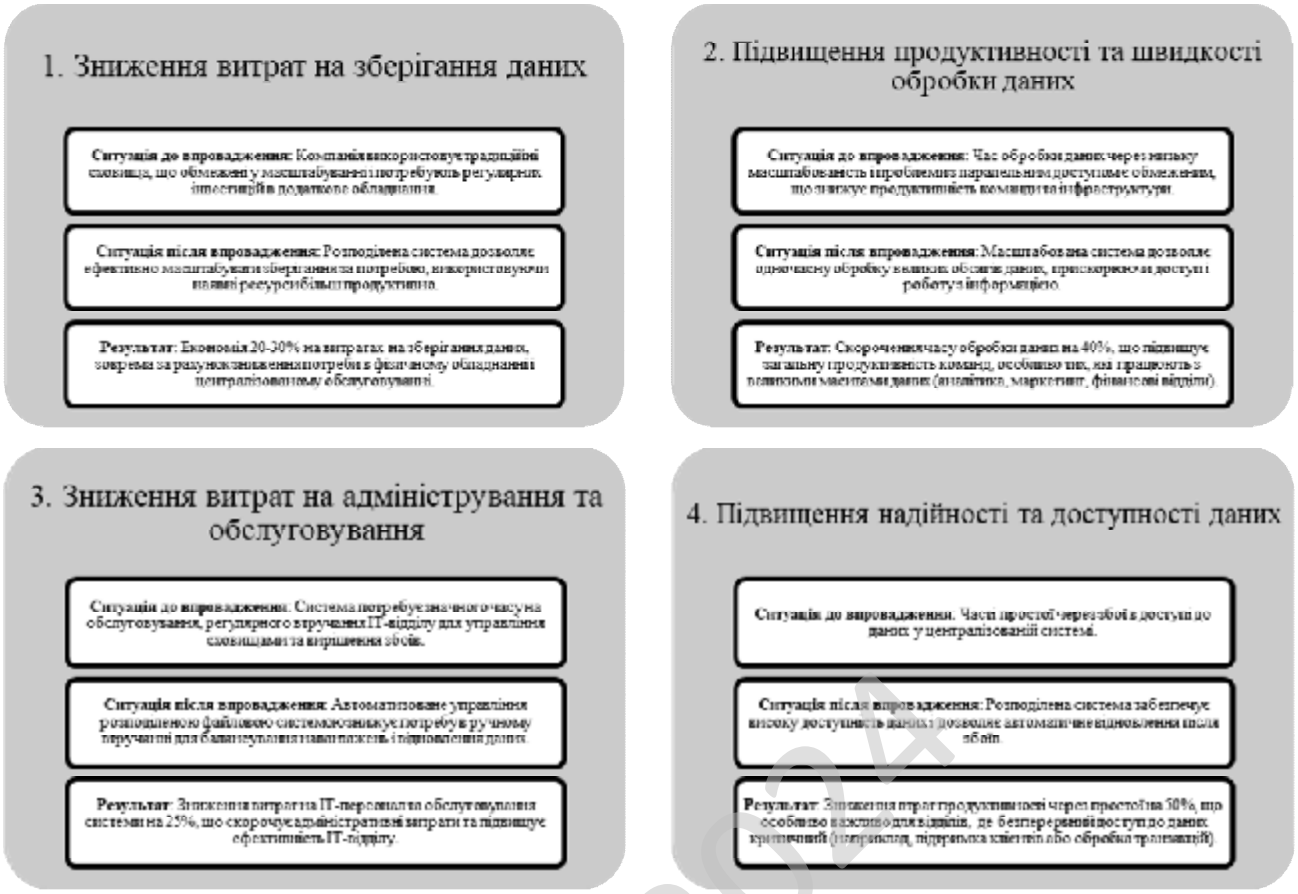

Рисунок 7.4 – Економічна ефективність від реалізації проєкту для клієнта

На основі цих показників можна оцінити загальний економічний ефект. Загальна економія витрат: близько 25-35% на зберіганні, обслуговуванні та адмініструванні. Підвищення продуктивності та скорочення часу на обробку: до 40%. Зниження втрат від простоїв: до 50%.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Алгоритм для просування проєкту програмної реалізації системи масштабованої розподіленої файлової системи запропоновано на рисунку 7.5. Запропонований підхід забезпечить послідовне зростання інтересу до проєкту, підвищить його впізнаваність та, відповідно, рівень продажів.

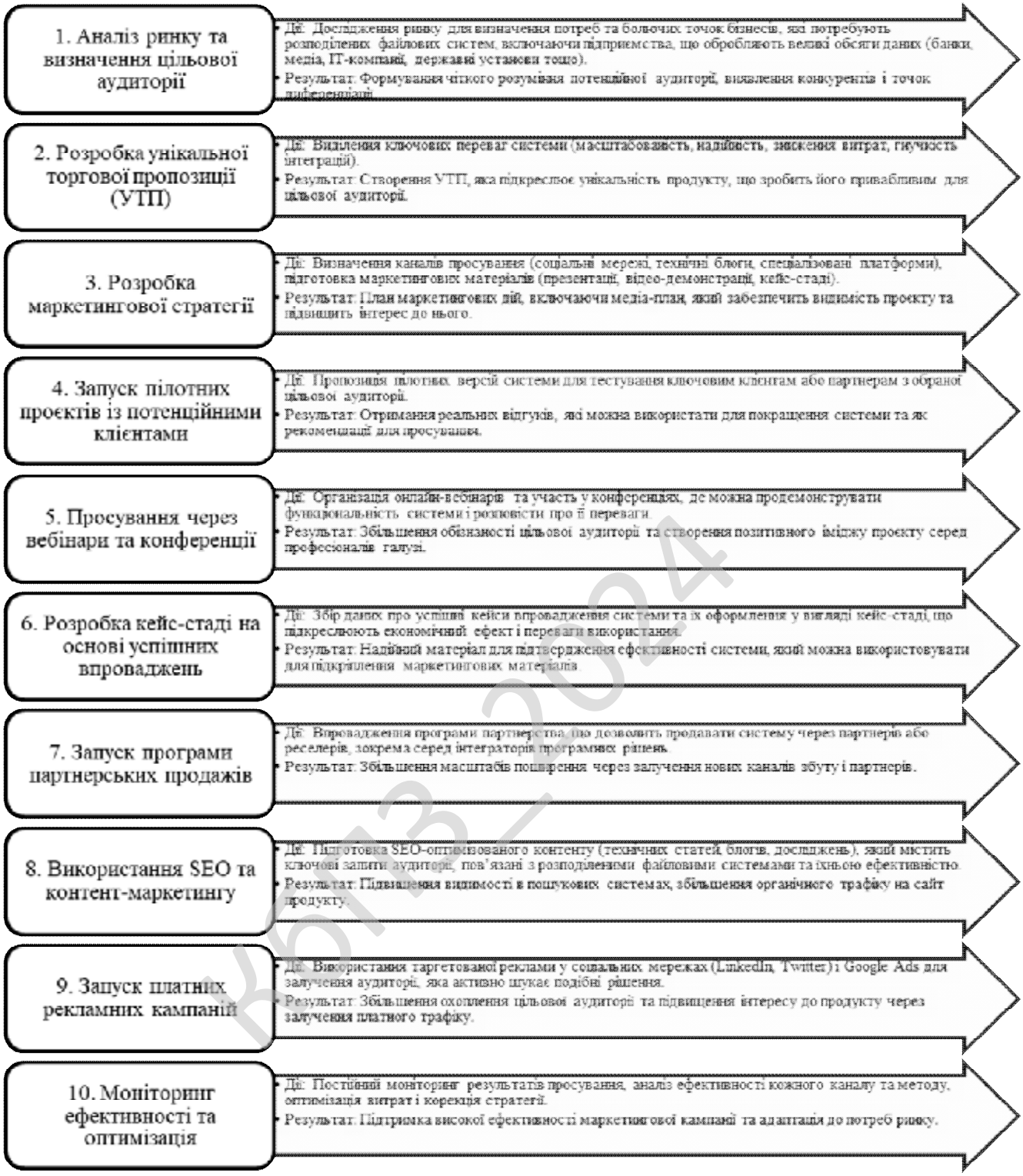

Рисунок 7.5 – Алгоритм для просування проєкту

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту та шляхів реалізації проєкту масштабованої розподіленої файлової системи можна використовувати комплексний підхід. Нижче наведено основні рекомендації щодо оптимізації, які допоможуть ефективно залучати клієнтів, розширювати канали збуту та збільшувати продажі:

– аналіз і сегментація цільової аудиторії;
– розширення каналів збуту через партнерські програми;
– впровадження багатоканальної стратегії продажу;
– використання SaaS-моделі для розширення ринку;
– автоматизація процесів продажу;
– залучення та підвищення лояльності клієнтів;
– оцінка ефективності та адаптація каналів збуту.

Цей комплексний підхід допоможе оптимізувати канали збуту та шляхи реалізації проєкту масштабованої розподіленої файлової системи, розширити охоплення аудиторії, а також покращити ефективність продажів і лояльність клієнтів.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключовими факторами успіху проєкту програмної реалізації системи масштабованої розподіленої файлової системи є кілька важливих аспектів, які забезпечують ефективність, конкурентоспроможність і довгострокову цінність для користувачів (рисунок 7.6). Загалом, для успіху проєкту важливо поєднувати якісну технічну реалізацію з орієнтацією на потреби клієнтів, конкурентними перевагами та постійним розвитком продукту.

Рисунок 7.6 – Ключові фактори успіху проєкту

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Законом України “Про охорону праці” [3] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [5], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби польців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робітників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [5], та «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення негативного впливу комп’ютера на організм людини визначимо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

8.2 Пожежна безпека

Пожежі в приміщеннях з оргтехнікою становлять особливу небезпеку, бо поєднані з великими матеріальними збитками. Пожежа може виникнути при взаємодії горючих речовин і джерел запалювання. Горючими речовинами є будівельні та опоряджувальні матеріали, пластмасові корпуси техніки, шнури тощо. Джерелами запалювання можуть бути електронні схеми комп'ютерів, принтерів, пристроїв електроживлення, де внаслідок різних порушень виникає перегрівання елементів, утворюються електричні іскри та дуги, здатні спричинити займання горючих матеріалів.

З метою виявлення початкової стадії займання необхідно використовувати пристрої систем автоматичного пожежогасіння там, де цього вимагають правила пожежної безпеки.

При обслуговуванні, ремонтних та профілактичних роботах використовуються різні легкозаймисті рідини, прокладаються тимчасові електропровідники, здійснюється паяння. Виникає додаткова пожежна небезпека, яка потребує відповідних заходів пожежного захисту. До засобів гасіння пожежі, призначених для локалізації невеликих займань, належать вогнегасники, сухий пісок, азбестові ковдри. Приміщення, в якому встановлено комп'ютери і де немає необхідності влаштування систем автоматичного пожежогасіння, необхідно

оснащувати переносними вуглекислотними з розрахунку 2 шт. на кожні 20 $м^2$ в приміщеннях. Звуковбирне облицювання стін, стель приміщень треба виконувати з негорючих та важко горючих матеріалів.

Електроустановки (можливість їх застосування, монтаж, накладка експлуатація) повинні відповідати вимогам чинних правил улаштування електроустановок, правил технічної експлуатації, електроустановок та інших нормативних документів.

Ймовірність виникнення пожежі від електротехнічного та іншого одиничного виробу не повинна перевищувати 10^{-6} на рік. При короткому замиканні в місцях з'єднання проводів опір практично дорівнює нулю, звідси величина струму досягає дуже великих значень.

Персональні комп'ютери після закінчення роботи повинні відключатися від мережі не рідше 1 разу на квартал, необхідно очищати від пилу агрегати та вузли, кабельні канали та простір між підлогами. Не дозволяється розміщувати комп'ютерні зали ЕОМ у підвалах; проводити ремонт вузлів (блоків) ЕОМ безпосередньо у залах, де знаходяться ПК (персональні комп'ютери), залишати без нагляду ввімкнену в мережу електронну апаратуру, яка використовується для контролю ЕОМ.

Електричний струм силою 0.1 А є небезпечним для людини. Для попередження травм усе електричне обладнання повинне бути заземлене. Приступаючи до роботи необхідно перевірити справність обладнання, ізоляцію проводів і надійність заземлення. Доторкання до оголених струмоведучих і незахищених частин в електроустаткуванні забороняється. В разі виявлення порушень ізоляції електропроводів, відкритих струмоведучих частин електроустаткування або порушення заземлення треба негайно повідомити про це свого начальника для вжиття заходів щодо усунення несправності. Проводити самому ремонт електроустаткування забороняється.

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

У зазначеному приміщенні працюють 4 людей.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	5,4
Довжина	6
Висота	2,75

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого·

Геометрична характеристика	Одиниця виміру	Нормативне значення *	Фактичне значення
Площа, S	$м^2$	не менше 6.0	8,1
Обсяг, V	$м^3$	не менше 20.0	24,3

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин) [2].

За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об’єм приміщення у розрахунку на одне робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2], але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [5] та НПАОП 0.00-1.28-10 «Правила охорони

праці під час експлуатації електронно-обчислювальних машин»). Таким чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами. У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура, °С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0.1	22-23	40-55	0.1
Тепла	23-25	50-70	0.1	24-25	50-65	0.11

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року. В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні. Крім того, офісні приміщення можуть бути додатково оснащені сучасними очищувачами повітря.

У приміщенні знаходяться наступні джерела шуму: принтер HP 1100, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці. З 2019 року діють Державні будівельні норми України “Природне і штучне освітлення” – ДБН В.2.5-28:2018 [1], у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних. Працю працівника, який постійно працює за комп’ютером, згідно ДБН В.2.5-28:2018 [1], можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк. [1], Крім того, все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

– розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значень.

Таким чином можна припустити, що основною причиною можливого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга).

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

Працівники повинні дотримуватися рекомендацій:

- яскравість монітору – не менше 100Kg/м^2;

– відношення яскравості монітора до яскравості оточуючих його поверхонь в робочій зоні – не більше 3:1;

– мінімальний розмір точки свічення не більше 0,4 мм для монохромного монітора і не менше 0,6 мм для кольорового, контрастність зображення знаку – не менше 0,8.

При виявленні будь-яких неполадок роботу не розпочинати, повідомити про це керівника.

Працівникам потрібно дотримуватися вимог безпеки під час виконання роботи:

– необхідно стійко розташовувати клавіатуру на робочому столі, не опускати її хитання. Під час роботи на клавіатурі сидіти прямо, не напружуватися;

– для забезпечення несприятливого впливу на користувача пристроїв типу ”миша” належить забезпечувати вільну велику поверхню столу для переміщення ”миші” і зручного упору ліктьового суглоба;

– не дозволяються сторонні розмови, подразнюючі шуми;

періодично при вимкненому комп'ютері прибирати ледь змоченою мильним розчином бавовняною ганчіркою порох з поверхонь апаратури. Екран ВДТ та захисний екран протирають ганчіркою, змоченою у спирті. Не дозволяється використовувати рідинні або аерозольні засоби чищення поверхонь комп'ютера.

8.5 Розрахункова частина

Для захисного штучного заземлення застосовуються вертикальні електроди: металевий куток 50·50·5 мм, довжиною L=3 м, та горизонтальний електрод – металева полоса з перетином 40·4 мм. Напруга, яка подається до будівлі – 220/380 В. Розрахункова схема розташування заземлюючих електродів – у ряд (рис. 8.1).

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару грунта – чорнозем, нижнього шару грунта – глина (питомий опір ρ_2 = 40 Ом·м). Умовна товщина верхнього шару грунта: H = 0.4 м. Відстань між вертикальними заземлювачами (електродами) A = 3 *м*. Глибина закладення горизонтального контура заземлення t=0.8 м. Опір заземлювача, який нормується: R_{3H} = 4 Ом.

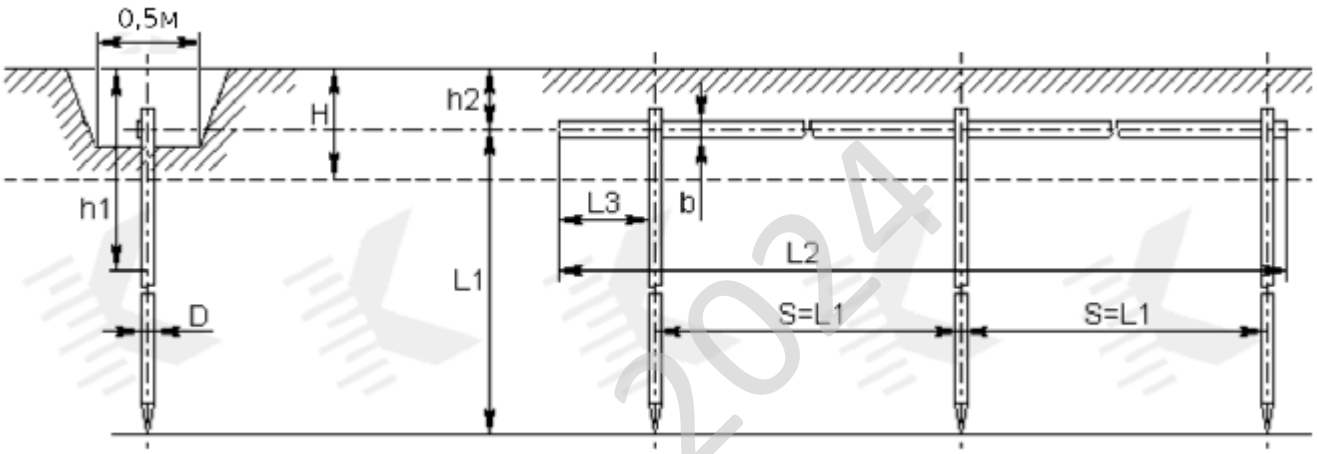

Рисунок 8.1 – Схема штучного заземлення

Необхідно визначити необхідну кількість вертикальних заземлювачів та довжину полоси (горизонтального заземлювача).

Проведемо розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0.8+3/2=2.3 \text{ м.}$$

Розрахунковий питомий опір грунта (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі грунта):

$$\rho = \psi \rho_2 = 1.36 \cdot 40 = 54.5 \text{ Ом·м.}$$

де ψ = 1.36 – табличне значення коефіцієнта сезонності для відповідної кліматичної зони у багатошаровому ґрунті [6];

ρ_2 = 40 Ом·м – табличне значення питомого опору нижнього шару ґрунту (глина) [11].

Еквівалентний діаметр вертикального електроду (кутика) [11]:

$$D_в = 0.95 \cdot К = 0.95 \cdot 50 = 47.5 \text{ мм} = 0.0475 \text{м}.$$

де К= 50 мм – розмір металевого кутика (заданий).

Опір розтіканню електричного струму одного електрода вертикального заземлювача (без врахуванням заглиблення заземлювача) [11]:

$$R_О = 0.366(\rho/L) \cdot \lg(4L/D_в) = 0.366 \cdot (54,5/3) \cdot \lg(4 \cdot 3/0.0475) = 15.9 \text{ Ом}.$$

Відношення A/L=3/3=1.

Опір розтіканню електричного струму одного електрода вертикального заземлювача з урахуванням заглиблення заземлювача [11]:

$$R_О = 0.366 \cdot (\rho/L) \cdot [\lg(2L/D_в) + (1/2) \cdot \lg((4t+L)/(4t-L))] =$$

$$= 0.366(54.5/3) \cdot [(\lg(2 \cdot 3/0.0475) + (1/2) \cdot \lg((4 \cdot 2.3+3)/(4 \cdot 2.3-3))] = 14.9 \text{ Ом}.$$

Визначаємо коефіцієнт екранування вертикальних електродів Кев=0.8 при орієнтовній кількості вертикальних електродів, яке дорівнює 4 [11].

Визначаємо необхідну кількість вертикальних заземлювачів (без врахування горизонтального заземлювача), при $R_{ЗН}$ = 4 Ом

$$N = R_О / (К_{ев} \cdot R_{зн}) = 14.9 / (0.8 \cdot 4) = 4.66 \approx 5 \text{ шт}.$$

Визначаємо довжину з'єднуючої полоси:

$$L_П = 1.05 \cdot A \cdot N = 1.05 \cdot 3 \cdot 4.66 = 14.6 \approx 15 \text{ м}.$$

Опір розтіканню електричного струму з'єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору грунта $К_П$ [11]:

$$R_П = 0.366 \cdot (\rho_2 \cdot К_П / L_П) \cdot \lg(2(L_П \cdot L_П)/(B \cdot t)) =$$

$$= 0.366 \cdot (40 \cdot 5/14.6) \cdot \lg((2 \cdot 14.6^2)/(0.04 \cdot 0.8)) = 20.5 \text{ Ом}.$$

де $К_П$ = 5 – табличне значення кліматичного коефіцієнта питомого опору грунта для відповідної кліматичної зони для з'єднуючої полоси [11]:

B = 40 мм = 0.04 м – ширина з'єднуючої полоси (задана).

Загальний опір розтіканню електричного струму заземлювача [11]:

$$R = (R_о \cdot R_п)/(R_о \cdot \eta_п + N \cdot R_п \cdot К_{ев}) =$$

$$= (14.9 \cdot 20.5)/(14.9 \cdot 0.75 + 4.66 \cdot 20.5 \cdot 0.8) = 3.5 \text{ Ом}.$$

де $\eta_п = 0{,}75$ – табличне значення коефіцієнта екранування з'єднуючої полоси [11].

Умова $R \leq R_{ЗН}$ виконується ($3.5 \leq 4$).

Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи масштабованої розподіленої файлової системи.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів масштабованої розподіленої файлової системи.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем масштабованої розподіленої файлової системи.

– Досліджена система масштабованої розподіленої файлової системи.

– На основі отриманих результатів досліджень створена програмна реалізація системи масштабованої розподіленої файлової системи.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання масштабованої розподіленої файлової системи.

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

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

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

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм RC5.

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

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лівітчук О.В. Дослідження та програмна реалізація системи масштабованої розподіленої файлової системи // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2024.

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

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

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

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

6. 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.

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

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

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

10. 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.

11. 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.

12. 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.

13. 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.

14. 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

15. 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.

16. 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.

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

18. 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.

19. 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.

20. 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.

21. 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.

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

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

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. 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.

34. 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.

35. 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.

36. 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.

37. 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).

38. 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.

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

40. 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.

41. 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.

42. Смірнов О.А., Коваленко О.В., Коваленко А.С., Смірнов С.А. Розробка методу передтестової компіляції й розподілу доступу. Збірник наукових праць III міжнародної науково-практичної конференції "Інформаційна безпека та комп'ютерні технології", м. Кропивницький. 19-20 квітня 2018р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215

43. Smirnov Oleksii, Kovalenko Oleksandr, Kovalenko Anna, Smirnov Serhii. Method of testing the DOM XSS vulnerability. International Conference «Information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces "Alexandru cel Bun". 2017. P7.

44. Смірнов О.А., Смірнов С.А., Коваленко О.В., Коваленко А.С. Технологія тестування DOM XSS уразливості. Науково-практичний журнал кібер безпеки (SPCSJ) № 1. [Електронний журнал]. Грузія. Тбілісі: SCSA – 2017.

45. Смірнов О.А., Лисенко І.А. Інформаційна технологія проектування тестових наборів з урахуванням вимог до програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 4 (44). – Полтава: ПолтНТУ. – 2017. – С. 112-115.

46. Смірнов О.А., Смірнов С.А., Рябой Д.К., Рябая О.В. Модель вузла комутації з відносними пріоритетами, резервуванням ресурсів і обліком реальної надійності обслуговуючих приладів .Збірник тез всеукраїнської науково-практичної інтернет-конференції «Автоматика та комп'ютерно-інтегровані технології у промисловості, телекомунікаціях, енергетиці та транспорті».

м. Кропивницький. 16-17 листопада 2017 р. – Кропивницький: ЦНТУ. – 2017. – С. 198-199.

47. Смірнов О.А., Коваленко О.В. Використання псевдобулевих методів бівалентного програмування для управління ризиками розробки програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ. – 2016. – С. 98-103.

48. Смірнов О.А., Лисенко І.А. Формалізація процесу проектування тестових наборів. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 3 (48). – Харків: ХУПС. – 2016. – С.96-100.

49. Смірнов О.А., Лисенко І.А. Удосконалення методу перевірки коректності таблиць рішень для подання тестових наборів. Збірник наукових праць "Системи обробки інформації". – Випуск 8 (145). – Х.: ХУПС – 2016. – С. 77-80.

50. Смірнов О.А., Лисенко І.А. Розробка впорядкованих каскадних таблиць рішень із використанням матриць слідування. Збірник наукових праць "Системи обробки інформації". – Випуск 6 (143). – Х.: ХУПС – 2016. – С. 216-220.

51. Смірнов О.А., Коваленко О.В., Якименко Н.М., Доренський О.П. Метод кількісної оцінки ризиків розроблення програмного забезпечення. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. – 2016. – С. 128-133.

52. Смірнов О.А., Коваленко О.В., Якименко Н.М., Доренський О.П. Метод якісного аналізу ризиків розроблення програмного забезпечення. Наука і техніка Збройних Сил України. – Випуск 2(23). – Харків: ХУПС. – 2016. – С. 150-158.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи масштабованої розподіленої файлової системи.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 21-13 від 07.08.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи масштабованої розподіленої файлової системи.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

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

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи масштабованої розподіленої файлової системи;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Visual C#.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Економічні вимоги

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

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

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий аналіз санітарно-гігієнічних умов праці на робочому місці програміста.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 99 аркушів.

10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Маркетингове та економічне обґрунтування ІТ-проєкту.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

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

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 17.12.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

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

_____ Коваленко А.С.

Дослідження та програмна реалізація системи масштабованої розподіленої файлової системи

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 56

Літера: РП

Кропивницький – 2024 року

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 = "Довжина коду";
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // redundancyMacTrackBar
            //
            this.redundancyMacTrackBar.BackColor =
System.Drawing.Color.Transparent;
            this.redundancyMacTrackBar.BorderColor =
System.Drawing.SystemColors.ActiveBorder;
            this.redundancyMacTrackBar.Font = new System.Drawing.Font("Verdana",
8.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.redundancyMacTrackBar.ForeColor =
System.Drawing.Color.FromArgb(((int)(((byte)(123)))), ((int)(((byte)(125)))),
((int)(((byte)(123)))));
            this.redundancyMacTrackBar.IndentHeight = 6;
```

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

```
            this.allVolCountMacTrackBar.ValueChanged += new
EConTech.Windows.MACUI.ValueChangedHandler(this.allVolCountMacTrackBar_ValueChan
ged);
            this.allVolCountMacTrackBar.MouseUp += new
System.Windows.Forms.MouseEventHandler(this.allVolCountMacTrackBar_MouseUp);
            //
            // browser
            //
            this.browser.AutoValidate =
System.Windows.Forms.AutoValidate.EnablePreventFocusChange;
            this.browser.ListViewMode = System.Windows.Forms.View.List;
            this.browser.Location = new System.Drawing.Point(12, 131);
            this.browser.Name = "browser";
            treeNode1.Name = "";
            treeNode1.Text = "";
            treeNode2.ImageIndex = 18;
            treeNode2.Name = "backreg";
            treeNode2.SelectedImageIndex = 20;
            treeNode2.Text = "backreg";
            treeNode3.Name = "";
            treeNode3.Text = "";
            treeNode4.ImageIndex = 18;
            treeNode4.Name = "";
            treeNode4.SelectedImageIndex = 20;
            treeNode4.Text = "BChH";
            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::RecoveryData.Properties.Resources.table_sql_view32451;
            this.testButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.testButton.Location = new System.Drawing.Point(111, 27);
            this.testButton.Name = "testButton";
            this.testButton.Size = new System.Drawing.Size(100, 97);
            this.testButton.TabIndex = 4;
            this.testButton.Text = "Перевірка цілісності";
            this.testButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.testButton.UseVisualStyleBackColor = true;
            this.testButton.Click += new
System.EventHandler(this.testButton_Click);
            //
            // repairButton
            //
            this.repairButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.repairButton.Image =
global::RecoveryData.Properties.Resources.medical_bag465471;
            this.repairButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.repairButton.Location = new System.Drawing.Point(210, 27);
            this.repairButton.Name = "repairButton";
            this.repairButton.Size = new System.Drawing.Size(100, 97);
            this.repairButton.TabIndex = 3;
            this.repairButton.Text = "Відновлення цілісності";
            this.repairButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.repairButton.UseVisualStyleBackColor = true;
            this.repairButton.Click += new
System.EventHandler(this.repairButton_Click);
            //
            // recoverButton
            //
            this.recoverButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.recoverButton.Image =
global::RecoveryData.Properties.Resources.redo6786987;
            this.recoverButton.ImageAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.recoverButton.Location = new System.Drawing.Point(309, 27);
            this.recoverButton.Name = "recoverButton";
            this.recoverButton.Size = new System.Drawing.Size(100, 97);
            this.recoverButton.TabIndex = 2;
            this.recoverButton.Text = "Читання даних з диску";
            this.recoverButton.TextAlign =
System.Drawing.ContentAlignment.BottomCenter;
            this.recoverButton.UseVisualStyleBackColor = true;
            this.recoverButton.Click += new
System.EventHandler(this.recoverButton_Click);
            //
```

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

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

        }

        #endregion

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

Файл ProcessForm.cs - вікно відображення процесів запису/читання та перевірки цілісності даних

```
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 якщо керуючі ресурси повинні бути
розташовані, у іншому випадку  false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

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

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

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

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

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

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

```
            //
            this.errorCountLabel_.AutoSize = true;
            this.errorCountLabel_.Location = new System.Drawing.Point(28, 41);
            this.errorCountLabel_.Name = "errorCountLabel_";
            this.errorCountLabel_.Size = new System.Drawing.Size(35, 13);
            this.errorCountLabel_.TabIndex = 0;
            this.errorCountLabel_.Text = "Error :";
            this.toolTip.SetToolTip(this.errorCountLabel_, "Лічильник некоректно
оброблених файлів");
            //
            // okCountLabel_
            //
            this.okCountLabel_.AutoSize = true;
            this.okCountLabel_.Location = new System.Drawing.Point(28, 20);
            this.okCountLabel_.Name = "okCountLabel_";
            this.okCountLabel_.Size = new System.Drawing.Size(28, 13);
            this.okCountLabel_.TabIndex = 0;
            this.okCountLabel_.Text = "OK :";
            this.toolTip.SetToolTip(this.okCountLabel_, "Лічильник коректно
оброблених файлів");
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // 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 BCHEncoder.cs - декодер БЧХ

```
using System;
using System.Threading;

namespace RecoveryDisk
{
    /// <summary>
    /// Клас кодера БЧХ
    /// </summary>
    public class BCHEncoder : BCHBase
    {
        #region Construction & Destruction

        /// <summary>
        /// Конструктор кодера за замовчуванням
        /// </summary>
        public BCHEncoder()
        {
            // 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 BCHEncoder(int dataCount, int eccCount)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, (int)BCHType.Alternative);

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Установка конфігурації кодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="codecType">Тип кодека кодера БЧХ (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)BCHType.Dispersal)
            {
                maxVolCount = (int)BCHConst.MaxVolCountDisp;

            } else
            {
                maxVolCount = (int)BCHConst.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.eBCHType)
                    )
                {
                    this.mainConfigChanged = true;
                }

                // Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eBCHType = 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.eBCHType == (int)BCHType.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.eBCHType == (int)BCHType.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.eBCHType == (int)BCHType.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 (OnBCHMatrixFormingFinish != null)
                {
                    //...повідомляємо, що екземпляр класу готовий до роботи
                    OnBCHMatrixFormingFinish();
                }

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

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

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

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

        #endregion Private Operations
    }
}
```

Файл BCHDecoder.cs - декодер БЧХ

```
using System;
using System.Threading;

namespace RecoveryDisk
{
    /// <summary>
    /// Клас декодера БЧХ
    /// </summary>
    public class BCHDecoder : BCHBase
    {
        #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 BCHDecoder()
        {
            // 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 BCHDecoder(int dataCount, int eccCount, int[] volList)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList, (int)BCHType.Alternative);

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

        /// <summary>
        /// Розширений конструктор декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        /// <param name="codecType">Тип кодера БЧХ (по типу матриці)</param>
        public BCHDecoder(int dataCount, int eccCount, int[] volList, int
codecType)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList, codecType);

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

```
        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Установка конфігурації декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        /// <param name="codecType">Тип кодека кодера БЧХ (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)BCHType.Dispersal)
            {
                maxVolCount = (int)BCHConst.MaxVolCountDisp;

            } else
            {
                maxVolCount = (int)BCHConst.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.eBCHType)
                    )
                {
                    this.mainConfigChanged = true;
                }

                // Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eBCHType = 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.eBCHType == (int)BCHType.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)
                     &&
                        (OnUpdateBCHMatrixFormingProgress != null)
                   )
                {
                    //...виводимо дані
                    OnUpdateBCHMatrixFormingProgress((((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.eBCHType == (int)BCHType.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.eBCHType == (int)BCHType.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 (OnBCHMatrixFormingFinish != null)
                {
                    //...повідомляємо, що екземпляр класу готовий до роботи
                    OnBCHMatrixFormingFinish();
                }
```

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

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

        #endregion Private Operations

        #region Public Properties

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

                } else
                {
                    return null;
                }
            }
        }

        #endregion Public Properties
    }
}
```

Файл FileAnalyzer.cs - контроль цілісності даних

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

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

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

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

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

        #endregion Delegates

        #region Public Properties & Data

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

                } else
                {
                    return false;
                }
            }
        }

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

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

        } else
        {
            return false;
        }
    }
}

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

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

        } else
        {
            return false;
        }
    }
}

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

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

        } else
        {
            return null;
        }
    }
}

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

/// <summary>
```

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

                } else
                {
                    return false;
                }
            }
        }

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

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

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

                            break;
                        }

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

                            break;
                        }

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

                            break;
                        }

                        case 3:
                        {
```

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

                            break;
                        }

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

                            break;
                        }
                    }

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

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

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

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

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

        #endregion Public Properties & Data

        #region Data

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

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

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

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

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

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

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

        /// <summary>
        /// Тип кодера БЧХ (по типу використовуваної матриці кодування)
        /// </summary>
        private int codecType;

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

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

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

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

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

        #endregion Data

        #region Construction & Destruction

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

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

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

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

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

            // Екземпляр класу повністю закінчив обробку?
```

```
            this.finished = true;

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

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

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

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Метод запуску потоку обробки обчислення й записи CRC64 у кінець
файлів
        /// </summary>
        /// <param name="path">Шлях до файлів для обробки</param>
        /// <param name="fileName">Ім'я файлу для обробки</param>
        /// <param name="dataCount">Конфігурація кількості основних
томів</param>
        /// <param name="eccCount">Конфігурація кількості томів для
відновлення</param>
        /// <param name="codecType">Тип кодера БЧХ (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)BCHConst.MaxVolCountAlt)
                )
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

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

                return false;
            }

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

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

            // Зберігаємо тип кодера БЧХ (по типу використовуваної матриці
кодування)
            this.codecType = codecType;

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

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

                // Повертаємо результат обробки
                return this.processedOK;
            }

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

            //...потім даємо йому ім'я...
            this.thrFileAnalyzer.Name = "FileAnalyzer.WriteCRC64()";

            //...установлюємо обраний пріоритет завдання...
```

```
            this.thrFileAnalyzer.Priority = this.threadPriority;

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

            // Повідомляємо, що все нормально
            return true;
        }

        /// <summary>
        /// Метод запуску потоку обробки перевірки CRC64, записаного в кінець
        /// кожного з файлів набору, з генеруванням списку наявних томів
"volList",
        /// який буде використаний декодером для відновлення даних
        /// </summary>
        /// <param name="path">Шлях до файлів для обробки</param>
        /// <param name="fileName">Ім'я файлу для обробки</param>
        /// <param name="dataCount">Конфігурація кількості основних
томів</param>
        /// <param name="eccCount">Конфігурація кількості томів для
відновлення</param>
        /// <param name="codecType">Тип кодера БЧХ (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)BCHConst.MaxVolCountAlt)
                )
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

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

                return false;
            }

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

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

            // Зберігаємо тип кодека кодера БЧХ (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 коректно (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 About.cs - вікно довідки про програму

```
namespace RecoveryData
{
    partial class AboutForm
    {
        /// <summary>
        /// Необхідні змінні розробника.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

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

        #region Windows Form Designer generated code

        /// <summary>
        /// Необхідний метод для підтримки розробника - не модифікується
        /// зміст цього метода використовується редактором коду.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.developersListBoxdevelopersListBox = new
System.Windows.Forms.ListBox();
            this.BCHIconTimer = 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[]
{
            "МАГІСТЕРСЬКА ДИПЛОМНА РОБОТА",
            "",
            "На тему:",
            "",
            "Дослідження та програмна реалізація системи масштабованої
розподіленої файлової системи",
            "",
            "",
            "Керівник: Коваленко А.С.",
            "",
            "Розробив: студент Лівітчук Олексій Віталійович",
            "                  гр. КІ 23 МЗ",
            "",
            "м. Кропивницький 2024"});
            this.developersListBoxdevelopersListBox.Location = new
System.Drawing.Point(11, 6);
            this.developersListBoxdevelopersListBox.Name =
"developersListBoxdevelopersListBox";
```

```
            this.developersListBoxdevelopersListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.developersListBoxdevelopersListBox.Size = new
System.Drawing.Size(330, 182);
            this.developersListBoxdevelopersListBox.TabIndex = 0;
            this.developersListBoxdevelopersListBox.TabStop = false;
            this.developersListBoxdevelopersListBox.SelectedIndexChanged += new
System.EventHandler(this.developersListBoxdevelopersListBox_SelectedIndexChanged
);
            //
            // BCHIconTimer
            //
            this.BCHIconTimer.Interval = 40;
            this.BCHIconTimer.Tick += new
System.EventHandler(this.BCHIconTimer_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 BCHIconTimer;
        private PinkieControls.ButtonXP okButtonXP;

    }
}
```