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

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2022 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-21М
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Вітряченко А.А.
«____» _____ 2022 р.

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Якименко Н.М.
«____» _____ 2022 р.
Рецензент
к.т.н., доцент
_____ Босько В.В.

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

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
« » _____ 2022 року

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

Вітряченко Аліні Андріївні
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища*

2. Керівник роботи *Якименко Наталія Миколаївна, кандидат фізико-математичних наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 19-13 від 17.08.2022 року

3. Строк подання студентом роботи до захисту *10.12.2022 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища*

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

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Економічна ефективність розробленої програми.*
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	Савеленко Г.В.	05.10.2022	14.11.2022
Охорона праці	Оришака О.В.	06.10.2022	16.11.2022

7. Дата видачі завдання « » 2022 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	10.10.2022 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2022 р.	
3.	Розробка моделі компонента	20.10.2022 р.	
4.	Розробка структур даних	25.10.2022 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2022 р.	
6.	Програмування алгоритмів	10.11.2022 р.	
7.	Розрахунок економічної ефективності	13.11.2022 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2022 р.	
9.	Оформлення ПЗ	17.11.2022 р.	
10.	Попередній захист роботи	10.12.2022 р.	

Дата видачі завдання
« » 2022 р.

Підпис керівника
_____ **Якименко Н.М.**
(прізвище та ініціали)

Завдання прийнято до виконання
« » 2022 р.

Підпис здобувача
_____ **Вітряченко А.А.**
(прізвище та ініціали)

АНОТАЦІЯ

Вітряченко А.А. Дослідження та програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для резервного копіювання даних хмарного, віртуального й фізичного середовища.

Метою роботи є дослідження та програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища.

Об'єктом дослідження є процес резервного копіювання даних хмарного, віртуального й фізичного середовища.

Предметом дослідження є методи резервного копіювання даних хмарного, віртуального й фізичного середовища.

Методи дослідження базуються на методах резервного копіювання даних, методах надійного зберігання даних й відновлення даних та методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища.

В процесі роботи над програмним забезпеченням виконано аналіз існуючих систем, апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з системою.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму реалізовано на мові Visual C# у середовищі .NET Framework.

Ключові слова: комп'ютерна інженерія, система резервного копіювання даних, хмарне середовище

ABSTRACT

Vitryachenko A.A. Research and software implementation of a cloud, virtual and physical environment data backup system. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2022.

In this final qualification work for the second (master's) level of higher education, software is developed, which is intended for backing up data in the cloud, virtual and physical environment.

The purpose of the work is research and software implementation of a system for backing up data in the cloud, virtual and physical environment.

The object of the study is the process of backing up data in the cloud, virtual and physical environment.

The subject of research is the methods of backing up data in the cloud, virtual and physical environment.

Research methods are based on data backup methods, reliable data storage and data recovery methods, and software development methods.

The result is a software implementation of a system for backing up data from the cloud, virtual and physical environment.

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

Developed user-friendly interface. Instructions for working with the system are given.

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

The program is implemented in Visual C # in the .NET Framework.

Keywords: computer engineering, data backup system, cloud environment

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

IT	-	Інформаційні технології – процеси, методи пошуку, збору, зберігання, обробки, надання, поширення інформації та способи здійснення таких процесів і методів.
RAID	-	Redundant Array of Independent Disks – технологія віртуалізації даних, яка об'єднує кілька дисків в логічний елемент.
ПК	-	Персональний комп’ютер
FTP	-	File Transfer Protocol – протокол передачі файлів по мережі
UDF	-	Universal Disk Format, універсальний дисковий формат – специфікація формату файлової системи, що не залежить від операційної системи
SAN	-	Storage Area Network - мережа зберігання даних (МЗД)
DHCP	-	Dynamic Host Configuration Protocol – протокол динамичної конфігурації вузла
HTTP	-	HyperText Transfer Protocol – протокол передачі гіпер тексту
IMAP	-	Internet Message Access Protocol – протокол доступу до електроної пошти Інтернету
ICMP	-	Internet Control Message Protocol – міжмережний протокол керуючих повідомлень
POP3	-	Post Office Protocol Version 3 – протокол поштового відділення, версія 3
SQL	-	Structured Query Language – мова структурованих запросів
SMTP	-	Simple Mail Transfer Protocol – простий протокол передачі пошти
SNMP	-	Simple Network Management Protocol – простий протокол керування мережею
UDP	-	User Datagram Protocol – протокол користувальницьких дейтаграм

ВСТУП

Актуальність теми. Застосування програмного забезпечення системи резервного копіювання даних відкривають широкі можливості для значного підвищення рівня автоматизації процесів роботи з даними в умовах нестачі часу, бюджету та ресурсів і тому дослідження і програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища є актуальною задачею в наш час.

Відомо, що для репутації компанії й взагалі для розвитку бізнесу важливими є дані. Щоб протистояти загрозам безпеки й кібератакам та зробити відновлення даних можливим, необхідно зробити захист інформації надійним.

Дані – це інформація, яка зберігається на вінчестері, на флеш-пристрої або в мережі. Втрата інформації, що несе велику цінність для користувача і на яку витрачено багато ресурсів через раптовий вихід з ладу носія буде катастрофою.

Можна перерахувати декілька причин збоїв:

– Пошкодження жорсткого диска ПК або флеш-пристрою, який містить інформацію. Термін придатності сучасних носіїв при активному застосуванні в середньому: флешпам’яті складає 5 років, жорсткого диску - від 5 до 10 років, в залежності від розміру накопичувача і умов використання. Пошкодження жорсткого диска або флеш-пристрою через обмежений ресурс може статися раптово і файли можуть бути втрачені назавжди.

– Збій, при якому система нездатна визначити внутрішню структуру диска та показати його вміст. Втрата документів повністю або частково через подібні збої файлової системи також можливі.

– Дія комп’ютерних вірусів. Наслідки вірусів можуть проявлятися по різному: від різних візуальних ефектів, що заважають працювати, до повної втрати інформації. Застосування антивірусних програм зменшує ймовірність

пошкоджень, та все ж таки, навіть регулярне оновлення антивірусної бази не дає повного захисту від нього.

– Небажані зміни користувачем. Спеціально чи випадково, можливо помилково внести зміни в документ.

– Нерідко зустрічається випадкове видалення файлів користувачем повз кошика і є необхідність їх відновити.

– Пожежа, землетрус, повінь та інші неприємності також можуть знищити необхідну інформацію.

Іноді знищені файли відновити взагалі не вдається. Втрата бази даних, знищення інформації, можливо навіть недоступність на деякий час цих даних може бути критичним для роботи підприємства.

Для запобігання втрати даних застосовують програмне забезпечення для резервного копіювання даних.

Мета й завдання дослідження. Метою роботи є розробка програмного забезпечення системи резервного копіювання даних хмарного, віртуального й фізичного середовища.

Для виконання завдань дослідження та досягнення мети роботи було описано порядок дій дослідження, що складається з наступних завдань:

– Огляд наявних систем резервного копіювання даних.

– Дослідження систем резервного копіювання даних.

– Програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища.

Практична цінність отриманих результатів дослідження та програмної реалізації системи резервного копіювання даних знаходиться в тому, що розглянуті алгоритми дозволяють розв’язувать задачі резервного копіювання даних досить успішно.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища, є актуальною задачею, яка потребує вирішення у даній

випускній кваліфікаційній магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Системи резервного копіювання даних хмарного, віртуального й фізичного середовища актуальні в даний час, оскільки втратити дані можна не тільки через віруси й хакерські атаки. Таке може статися внаслідок помилок користувачів і адміністраторів мережі, несправність устаткування, форс-мажорних обставин (крадіжок, пожеж, стихійних лих). Для захисту від цього призначені системи резервного копіювання та відновлення даних (Backup and Recovery). Вони із заданою періодичністю копіюють певну системними адміністраторами інформацію на резервні носії або в хмару. Резервувати можна як конкретні файли і папки, так і образи систем і серверів, вміст баз даних і додатків.

Створення резервних копій необхідно для відносно швидкого і незатратного відновлення інформації (документів, програм, налаштувань, фотографій та інше) у разі втрати робочої копії (оригіналу) інформації з якої-небудь причини. Крім цього, вирішується проблема передачі даних і роботи з загальними документами.

Система резервного копіювання та відновлення даних, що розв’язує три розповсюджені задачі для захисту даних малого бізнесу, які пов'язані з резервним копіюванням. Це проблеми обмеженого функціоналу, складність та висока вартість.

1.2 Область застосування

За допомогою систем резервного копіювання можливо забезпечити функціонування процесів бізнесу без перерв, а також захистити дані компанії від

втрат, викрадень та катастроф. Ці технології активно використовуються в IT-інфраструктурах організацій різних галузей і масштабів. Резервне копіювання даних – процес створення копії даних на носії, призначеному для відновлення даних в оригінальному місці їх розташування в разі їх пошкодження або руйнування. Крім того, система резервного копіювання – це один з необхідних методів забезпечення безперервності бізнесу. Створення централізованої системи для резервного копіювання дає можливість зменшити сукупну вартість застосованої IT-інфраструктури. Використання пристроїв резервного копіювання в централізованих системах оптимальніше, також скорочуються витрати на адміністрування в порівнянні з децентралізованими системами.

Наступні завдання системи визначаються з цілей резервного копіювання:

– Виділення цільових даних.

– Збереження зазначених даних для подальшого відновлення.

– Відновлення даних, що необхідно зберегти.

– Забезпечення стійкості до знищення та змін даних, що зберігаються.

– Розмежування доступу до збережених даних.

– Забезпечення контролю системи та процесу резервного копіювання.

Хмарні рішення підприємствам потрібно впевнено і швидко застосовувати для того, щоб отримати конкурентні переваги у розв’язанні різних завдань (наприклад, Microsoft Azure, AWS або Office 365). Та все ж таки впровадження хмарних систем потребує дій, що сплановані на захист даних. Якщо використовувати хмару не сплановано, можна зробити конфіденційні дані вразливими й бути впевненим в їх захисті при цьому.

Таким чином, виходячи з вищеперерахованого, програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища є актуальною задачею, яка потребує розв’язання у даній випускній кваліфікаційній магістерській роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Для розробки програмного забезпечення системи резервного копіювання даних хмарного, віртуального й фізичного середовища було розглянуто велику кількість програм для резервного копіювання та відновлення системи, включаючи утиліти, що дозволяють створювати бекап-копії жорстких дисків і логічних розділів, які сьогодні є найбільш затребуваними у користувачів. Ретельний опис деяких з них розібрано [1], з огляду на рівень популярності й складності їх використання. Наступні утиліти відомі користувачам, що користуються резервним копіюванням:

– Acronis True Image.

– Norton Ghost.

– Back2zip.

– Comodo BackUp.

– Backup4all.

– ABC Backup Pro.

– Active Backup Expert Pro.

– ApBackUP.

– File Backup Watcher Free.

– The Copier.

– Auto Backup і багато інших.

Acronis True Image

Цей засіб – одна з найпотужніших і популярних утиліт, що користується заслуженим успіхом і довірою багатьох користувачів. Відноситься до програм початкового рівня, але засіб має корисні потрібні можливості [2].

Після запуску програми користувач потрапляє до основного меню, де можна вибрати кілька варіантів дій. Один з варіантів дій є розділ, що призначений для створення резервних копій та можливість відновлення. Після початку роботи викликається майстер, що створює бекап. Таким чином вибираються об'єкти для копії, види, налагодження та інше для резервного копіювання.

Поєднання схем резервного копіювання та планувальника допомагає створити власну стратегію резервного копіювання. Схеми резервного копіювання дозволяють оптимізувати використання простору сховища резервних копій, підвищити надійність зберігання даних й автоматично видаляти застарілі версії резервної копії.

Резервне копіювання інформації має визначення деяких означень:

– Метод, який використовуються при створенні версій резервної копії (диференційне, інкрементне або повне резервне копіювання).

– Послідовність версій резервної копії, створених з використанням різних методів.

– Правила очищення версій.

Інкрементне резервне копіювання корисне тим, що буде допомога в заощадженні місця. Але якщо місця багато, можна застосувати повне резервне копіювання.

Переваги: утиліта показує досить високі показники по швидкості створення бекап-копії, часу, стиску. Так, наприклад, на стиснення даних близько 20 Гб знадобиться в середньому 8-9 хвилин, а розмір кінцевої копії скласти трохи більше 8 Гб.

Norton Ghost

Це теж потужна утиліта [3]. Після старту програми запускається "Майстер", що допомагає пройти всі кроки.

Дана утиліта примітна тим, що з її допомогою можна створити на вінчестері прихований розділ, де буде зберігатися копія (причому з неї можна відновити і дані, і систему). Крім того, в ній можна змінювати безліч параметрів:

тип точки відновлення, контроль зчитування, тип запису, стиснення, число точок для одночасного доступу й т. ін.

Налаштувавши розклад резервного копіювання, є можливість забезпечити автоматичне збереження внесених змін. При необхідності резервне копіювання можна запустити вручну. Крім того, продукт Norton Ghost підтримує запуск резервного копіювання в відповідь на задані події. Тобто, старт резервного копіювання може відбуватися з вказаним додатком. Також резервне копіювання може відбуватись за умови додавання на диск визначеного розміру інформації. У випадку збоїв в роботі комп'ютера інформацію можна відновити. Якщо відбувається відновлення, то робочий стан операційної системи, застосунків та файлів даних повертається.

Для роботи продукту Norton Ghost потрібно Microsoft .NET Framework 2.0 або більш пізньої версії. Якщо середовище .NET не встановлено, то вам буде запропоновано встановити його після завершення установки продукту Norton Ghost і перезавантаження комп'ютера.

Що ж стосується продуктивності, то 20 Гб додаток стискає до розміру трохи більше 7, 5 Гб, що за часом займає близько 9 хвилин. Загалом, результат досить непоганий.

Back2zip

Це програма резервного копіювання за розкладом. Максимально проста безкоштовна програма [4]. Вона може стати в пригоді всім, хто працює з документами на ПК і не хоче одного разу втратити цінну інформацію. Програма виконує копіювання даних з підтримкою zip-архівування. Вказано, що зберігання старих файлів – до 2 тижнів. Для виконання копіювання потрібно вибрати теку, що підлягає обробці й вказати місце, де буде зберігатися збережена інформація. Простий інтерфейс для користувача, вказується періодичність копіювання даних з інтервалом від 20 хвилин до 6 годин. Також можливо вибрати час, коли працює програма.

Вона відрізняється тим, що її інсталяція займає малий час (пару секунд).

Після запуску вона автоматично створює нове завдання і починає копіювання даних, це є головним недоліком програми.

Comodo BackUp

Ця утиліта здатна конкурувати навіть з комерційними продуктами. Вона особлива тим, що є наявність п'яти режимів роботи та можливість багатьох налаштувань [5].

Comodo BackUp це потужна і проста у використанні програма, яка допомагає домашнім і бізнес-користувачам захистити свої важливі дані від пошкодження або втрати. Додаток відрізняється простим інтерфейсом і розширеними можливостями.

Серед переваг можна відмітити, що утиліта здатна реагувати на зміни в файлах, що входять до складу резервної копії, в режимі реального часу. Як тільки вихідний файл змінюється і зберігається, додаток зразу створює його копію, додаючи й заміщаючи кінцевий елемент в бекапі. Не кажучи про планувальника, окремо можна відзначити початок створення копій або в момент старту, або при виході.

Backup4all

Це безкоштовна утиліта, що дозволяє одним натиском зробити резервні копії для всього, що може знадобитися надалі одночасно.

Backup4all дозволяє зберігати копії на різних зовнішніх та внутрішніх носіях таких як: DVD, CD, Blu-ray, HD-DVD, USB дисках й ін., також в мережах, або на FTP-серверах. Утиліта має досить багато редагованих параметрів і налаштувань [1], серед яких можна виділити чотири методи копіювання, а також підтримку файлової системи UDF. Крім того, інтерфейс простий, а показ тек і завдань представлено у вигляді дерева за типом провідника. Також користувач може розділяти дані, що копіюються, за категоріями зразок документів, малюнків і т. ін., привласнювати кожному проєкту власний ярлик. Є "Планувальник завдань", в якому можна вказати, наприклад, створення копій виключно в момент низького навантаження на процесор.

Backup4all має безліч корисних функцій і інтуїтивно-зрозумілий інтерфейс, що дозволяє як новачкам, так і досвідченим користувачам користуватися всіма функціями програми.

Рішення для серверних систем

Спеціальні програми для операцій резервного копіювання інформації є для серверних систем і мереж. Їх немало. Серед усього цього розмаїття можна виділити три найпотужніших:

– Symantec Backup Exec 11d System Recovery.

– Yosemite Backup Standard Master Server.

– Shadow Protect Small Business Server Edition.

Подібні утиліти вважаються добрим інструментом резервного копіювання для малого бізнесу. При цьому відновлення з нуля може здійснюватися з будь-якої робочої станції, що знаходиться в мережі. Але найголовніше, резервування потрібно зробити лише один раз, усі наступні зміни зберігатимуться автоматично. Всі програми мають інтерфейс типу «Провідник» і підтримують віддалене керування з будь-якого терміналу в мережі.

Для зберігання резервних копій використовуються різні носії. Найчастіше – жорсткі й твердотілі диски (HDD і SSD) в складі різних спеціалізованих пристроїв (NAS, RAID-масивів і т. ін.)[6].

RAID (Redundant Array of Independent Disks) – технологія віртуалізації даних, яка об'єднує кілька дисків в логічний елемент для надійності збереження інформації та підвищення продуктивності запам'ятовувальних пристроїв.

Дискові масиви можна використовувати для надійного зберігання інформації, щоб запобігти втраті даних у разі збою диска. Залежно від рівня RAID-масиву дані на дисках або дублюються, або рівномірно розподіляються. Використання мережі зберігання [7] (МЗД або Storage Area Network (SAN)) дозволяє повністю конвертувати резервний трафік з LCM на MZD.

МЗД — це архітектурне рішення для підключення до серверів зовнішніх накопичувачів даних (дискових масивів, оптичних приводів тощо). Це робиться

для того, щоб операційна система розпізнала підключені пристрої як локальні. Побудова мережі SAN розв'язує проблеми зниження загальної вартості мати систему зберігання даних, а також надає інструменти для організації надійного зберігання інформації [7].

Серед переваг архітектури МЗД є відсутність сильної прив'язки частин, що складають систему, до конкретних пристроїв зберігання даних[7]. Ця корисна функція була закладена в основу безсерверної технології резервного копіювання даних. У цій ситуації як сервер даних, так і пристрої, залучені до копіювання, можуть мати прямий доступ до дискового масиву. При резервному копіюванні інформації за блоками, що належать певному файлу, список номерів блоків, що належать до нього (або індекс), стоїть першим. Тому в подальшому є можливість підключити зовнішній пристрій під час резервного копіювання. Тому безсерверна операція копіювання може здійснювати пряме передавання даних між дисковими масивами та бібліотеками, підключеними до мережі зберігання даних. Для корпоративних мереж ця операція копіювання вважається чудовою, оскільки вони потребують безперервної роботи 24/7. Перевагою є те, що інформація передається через мережу МЗД і не навантажує сервери чи мережу.

Тиражування даних. Сучасні дискові сховища мають можливість створювати копії всередині власне масиву. Дані, створені за допомогою таких засобів, називаються Point-In-Time (PIT), тобто фіксованими на певний момент часу. Існує два типи засобів виготовлення копій PIT: клонування та «моментальний знімок» або snapshot. Клонування зазвичай означає повне копіювання даних. Для цього потрібен такий самий обсяг дискового простору, як і для вихідних даних, і певний час. При використанні такої копії немає навантаження на томи диска, що містять вихідні дані. Це означає, що дискова підсистема робочого сервера не перевантажується. Принцип дії snapshot інший. Це може бути реалізовано апаратно всередині масиву та програмно на продуктивному робочому сервері [7]. Коли починається резервне копіювання, програма-агент видає програмі команду. Усі транзакції завершено, а кеш

збережено на диску. Після цього створюється віртуальна структура - знімок, що представляє собою карту розташування блоків даних, яку ОС та інше програмне забезпечення сприймає як логічний том. Додаток на короткий час перериває стандартний режим роботи, необхідний для збереження даних. Потім програма продовжує працювати в стандартному режимі та змінювати блоки даних, при цьому старі дані блоку копіюються в область кешу знімків за допомогою драйвера знімків перед зміною, а нове розташування блоку вказується на карті розташування блоку даних. Таким чином, карта моментальних знімків завжди вказує на блоки даних, отримані під час виконання транзакцій програмою. Блоки даних, в які не вносилися зміни, зберігаються в попередньому місці, а старі дані змінених блоків - в області кешу зображень snapshot.

Застосування «моментальних знімків» для створення копій є економним для дискового простору, але має недолік. Це додаткове навантаження на дискову підсистему продуктивного сервера [7]. Метод резервного копіювання на певний момент Point-In-Time вибирається на етапі проектування системи резервного копіювання даних на основі бізнес-вимог до системи.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для впровадження системи резервного копіювання даних у хмарі, віртуальному та фізичному середовищі було розглянуто велику кількість ресурсів, технологій та мов програмування. Сучасні програмні комплекси дозволяють не тільки створювати резервні копії, а й здійснювати їх обслуговування. Мінімізувати кількість операцій, які необхідно виконувати вручну, можна за допомогою інтерактивних модулів – майстрів налаштувань. Вони дозволяють створювати дублікати (наприклад, у хмарі), дозволяють автоматизувати всі можливі дії. Вони вже записали шаблони завдань і типові конфігурації. Залишається лише вибрати їх і виконати налаштування з

урахуванням потреб, ресурсів і специфіки роботи компанії.

Ключовим моментом у розвитку сучасних засобів резервного копіювання є впровадження хмарних технологій [8]. Використання цих технологій полегшує керування резервними копіями та забезпечує постійний доступ до них. Клієнт може використовувати копії будь-де - вдома, на роботі, у будь-яких поїздках. За останні кілька років хмарні технології отримали величезний розвиток і набули величезної популярності завдяки появі безлімітного інтернету і розвитку мобільних мереж.

Високошвидкісний мобільний доступ до хмарних сховищ розширює їх можливості, хмарні копії тепер дозволяють спільно працювати над будь-яким проектом і служать зручним способом передачі файлів. Швидкість роботи з копіями в хмарі порівнюється зі швидкістю роботи на локальних носіях.

Сучасні центри обробки даних (ЦОД) досягають рівня надійності, вищого, ніж власні дискові масиви. Це досягається завдяки технологічним рішенням, які практично неможливо реалізувати в менших масштабах. У самих дата-центрах резервне копіювання робиться щодня, а то й кілька разів на день. Довгий час цілісність файлів резервної копії перевірялася лише одним способом: шляхом вилучення файлів і порівняння контрольних сум. Тож тестування займає майже стільки ж часу, скільки й створення копії. Крім того, перевірку неможливо автоматизувати, оскільки під час створення копії використовується шифрування. Захист паролем у налаштуваннях не ввімкнено політикою безпеки, тому пароль для розширення адміністратора потрібно щоразу вводити вручну. Новий підхід дозволяє поєднувати перевірку даних і шифрування. Він заснований на використанні метаданих. Під час створення копій створюються окремі контрольні блоки для файлів, які не потребують шифрування, оскільки вони не розкривають вміст вихідних даних. Цілісність файлу перевіряється за цим блоком керування, поки вміст не витягується. Перевагою є те, що перевірка цих даних може здійснюватися автоматично, а час перевірки скорочується в кілька разів. Однак цей підхід має і свої недоліки, які проявляються в меншій надійності.

Хмарне зберігання даних вимагає менше грошей для його підтримки. З використанням хмарних технологій користувачеві більше не потрібно підраховувати вільний простір на диску, що, безсумнівно, є великою перевагою і дає можливість працювати над чимось іншим, наприклад, над оптимальними методами захисту своїх даних.

Тому перевірка копій є обов'язковим кроком, який гарантує їх доступність і відсутність пошкоджених резервних копій. Зберігання даних за допомогою хмарних технологій дає наступні переваги:

1. Підвищує доступність даних.

2. Забезпечує необмежену пам'ять.

3. Звільняє власні локальні диски.

4. Зменшує витрати на резервне копіювання.

5. Не потребує витрат на збереження резервних копій.

6. Зберігає копії за межами офісу та залишатиметься недоторканим у разі непередбачених обставин.

7. Дозволяє легко передавати великі файли.

8. Надає доступ до копій третім особам за закритим посиланням тощо.

Використання хмарних технологій для зберігання даних може мати такі недоліки:

1. Збільшення трафіку.

2. Немає доступу до копій, якщо немає доступу до Інтернету.

3. Більша складність контролю доступу до файлів.

4. Підвищені вимоги до шифрування резервних копій.

5. Дострокове продовження передплати тощо.

IBM Cloud пропонує інфраструктурні рішення, платформні рішення та програмні рішення для розвитку бізнесу.

Інфраструктура як послуга (IaaS). Відповідальність компанії починається на рівні операційної системи, а доступність і надійність наданої інфраструктури забезпечується постачальником. Існує кілька випадків використання, коли ця

модель обслуговування може бути корисною. Компанії, які не мають власного центру обробки даних, розглядають IaaS як швидку та дешеву інфраструктуру для своїх бізнес-проектів, яку можна розширити або припинити за потреби. Іншим прикладом використання IaaS є традиційні компанії, яким потрібна обчислювальна потужність для обробки змінного робочого навантаження з меншими капітальними витратами. В обох випадках компанії платитимуть лише за послуги, якими вони користуються.

Платформа як послуга (PaaS). Для компаній-розробників, які хотіли б запровадити гнучку методологію розробки, модель PaaS є найбільш підходящою. Провайдери PaaS надають багато послуг, які можна використовувати в програмах. Ці послуги завжди будуть доступними та актуальними. PaaS забезпечує дуже простий спосіб тестування та створення прототипів нових програм. Це дозволяє економити кошти при розробці нових сервісів і програм. Програми можуть випускатися швидше, ніж зазвичай, щоб отримати відгуки користувачів.

Програмне забезпечення як послуга (SaaS). Сьогодні моделі надання послуг SaaS широко застосовуються багатьма компаніями, які хотіли б отримати вигоду від використання програм без необхідності підтримувати й оновлювати інфраструктуру та компоненти. Пошта, керування корпоративними ресурсами (ERP), спільна робота та офісні програми є найпопулярнішими рішеннями SaaS. Основні переваги моделі SaaS — гнучкість і еластичність. Універсального підходу до впровадження хмарних технологій не існує. Компанії повинні оцінити власні витрати та вигоди, а потім вирішити, яка модель найкраща. Кожна програма, потреба та процес є робочим навантаженням, і компанії, які вирішують перейти до хмари, зазвичай проводять поглиблену оцінку робочого навантаження.

Системи резервного копіювання та відновлення даних пропонують користувачам широкий спектр можливостей. Серед основних – підтримка різних середовищ і пристроїв, інтеграція з хмарними сервісами, швидкість роботи,

економічність зберігання даних, а також зручність користувача. Незважаючи на спільні пріоритети та схожий набір функцій, кожна з цих систем має свої сильні сторони, які можуть бути корисними для одних компаній, але водночас залишаються невибагливими для інших.

Більшість сьогоднішніх відомих і широко використовуваних утиліт в основному використовують принципи створення зображень і стиснення скопійованих даних. При цьому образи найчастіше використовуються саме для створення копій операційної системи, що дозволяє її подальше відновлення після непередбаченого критичного збою, а засоби для копіювання розділів диска або призначених для користувача файлів припускають стиснення за типом архівування.

Veeam [9][6] пропонує єдину платформу для резервного копіювання даних із хмарних, віртуальних і фізичних середовищ. Вона забезпечує резервне копіювання та відновлення для VMware vSphere, Microsoft Hyper-V, Nutanix AHV, фізичних серверів Windows і Linux, NAS, AWS, Azure, Office 365 тощо.

Запропоновані рішення для захисту даних малого бізнесу вирішують три найпоширеніші проблеми, пов’язані з резервним копіюванням: обмежена функціональність, складність і висока вартість. Надійне резервне копіювання та швидке відновлення усуває ризик втрати даних і вимушеного простою. Інноваційні рішення допомагають вирішити серйозні проблеми резервного копіювання. Проста установка та управління не вимагає спеціальної підготовки, великої кількості персоналу або складного навчання.

Переваги рішення: інтуїтивно зрозуміле резервне копіювання та миттєве відновлення даних з урахуванням статусу програми. Автоматичне тестування резервного копіювання з гарантованим відновленням даних для малого бізнесу. Повністю програмне рішення з вільним вибором апаратної платформи. Щоб отримати конкурентну перевагу, компаніям потрібно швидко вибирати та впевнено використовувати правильні хмарні рішення для різних завдань (наприклад, Microsoft Azure, AWS або Office 365).

Пропонуються локальні та інтегровані хмарні рішення, які забезпечують: захист і переносимість даних і додатків на будь-якій хмарній, віртуальній або фізичній платформі. Використання економічного хмарного сховища з рекомендаціями щодо вибору оптимальної стратегії зберігання та архівування даних малого бізнесу. Можливість перенесення ліцензій у міру зростання та змін віртуального середовища. Пропонуються вбудовані інтелектуальні функції безпеки та відповідності, щоб зменшити втрату даних. Veeam Backup Essentials забезпечує резервне копіювання та відновлення даних, а також моніторинг і звітування для фізичних, хмарних і віртуальних систем, включаючи VMware, Hyper-V і Nutanix AHV! Це економічно ефективне рішення пропонує широкий вибір варіантів за ціною, яка відповідає потребам малого бізнесу.

Політика збереження є ключовою частиною будь-якого резервного плану. Рано чи пізно доводиться видаляти точки відновлення, щоб звільнити місце для нових даних. Політика збереження визначає, що і коли видаляти.

Найпоширенішими типами політик зберігання є GFS і ToH.

GFS (Grandfather-Father-Son) – щоденне резервне копіювання «син», тижневе резервне копіювання «батько» і щомісячне резервне копіювання «дідусь». Якщо ви застосовуєте цю політику достатньо довго, рано чи пізно у вас закінчиться місце для зберігання, і вам доведеться щось видалити.

ToH (Ханойська вежа) - це дитяча гра, в якій вам потрібно переміщати піраміду з дисків з однієї смуги на іншу. Одночасно можна переміщати лише один диск, і заборонено класти більший диск на менший. Послідовність ходів для вирішення цієї головоломки є двійковим алгоритмом. Політика ToH дозволяє повторно використовувати простір для зберігання та розміщувати резервні копії на різних носіях. Це пояснюється тим, що якщо всі щоденні, щотижневі та щомісячні резервні копії зберігаються на одному диску, то в разі загального збою системи та диска всі дані будуть втрачені. На практиці цей вид політики використовується рідше через її складність.

Розглянемо системні вимоги до програмного забезпечення.

Виходячи з вищесказаного, можна сформулювати перелік функціональних і нефункціональних вимог до системи:

- Створення резервних копій необхідно виконати:

1) за графіком;

2) за запитом.

- Необхідно забезпечити резервне копіювання даних принаймні раз на день протягом 1 місяця.

- Час відновлення даних не повинен перевищувати 2 хвилини на 1 ГБ (обмеження пропускної здатності мережі - 81 секунда на 1 ГБ). Пропускна здатність мережі та швидкість диска вважаються обмежуючими факторами.

- Усі канали мають бути зашифровані. Можуть зберігатися конфіденційні дані, наприклад дані авторизації в онлайн-сервісах.

- Повинна бути можливість шифрувати резервні копії користувача за допомогою закритого ключа на комп'ютері користувача без дешифрування на сервері. Це важливо, оскільки користувачі можуть взагалі не довіряти серверу.

- Часткова копія не повинна займати багато часу, наприклад більше 30 хвилин. Для копіювання даних з ноутбука використовується Інтернет.

- Повинна бути потенційна можливість реплікації копій у хмару (можливість гнучкого налаштування реплікації для конкретного хмарного сховища) із шифруванням бекапів.

- Має бути просте централізоване управління з інтеграцією веб-інтерфейсу.

Для розробки програмного забезпечення системи резервного копіювання розглядалася мова Visual C#. Visual C# — це строго типізована об'єктно-орієнтована мова, призначена для розробки різноманітних безпечних і потужних програм, які працюють у .NET Framework. Ви можете використовувати Visual C# для розробки стандартних клієнтських програм Windows, веб-сервісів XML, розподілених компонентів, додатки «сервер-клієнт», баз даних і багато іншого. Visual C# 2012 має розширений редактор коду, зручний інтерфейс користувача,

вбудований налагоджувач і багато інших інструментів, призначених для спрощення розробки програм у Visual C# версії 5.0 і .NET Framework версії 4.5. Синтаксис Visual C# дуже виразний, але його легко вивчити. Кожен, хто знайомий із C, C++ або Java, легко впізнає синтаксис Visual C# за допомогою складних дужок. Розробники, які знають кожну з цих мов, як правило, зможуть ефективно працювати з Visual C# за дуже короткий час. Синтаксис Visual C# спрощує те, що було складно в C++, і надає такі потужні функції, як типи значень із підтримкою Null, перерахування, делегати, лямбда-вирази та прямий доступ до пам'яті, яких немає у Java. Visual C# підтримує універсальні методи та типи, які забезпечують вищий рівень безпеки та продуктивності, а також ітератори, які дозволяють реалізаціям колекції класів визначати власну ітераційну поведінку, яку можна легко використовувати в коді клієнта. Вирази інтегрованого запиту (LINQ) у Visual C# 5.0 роблять запит сильного типу першокласною мовною конструкцією.

Visual C# як об'єктно-орієнтована мова підтримує концепції інкапсуляції, успадкування та поліморфізму. Усі змінні та методи, включаючи метод Main — точку входу програми — інкапсульовані у визначеннях класу. Клас може успадковувати безпосередньо від одного батьківського класу, але може реалізовувати будь-яку кількість інтерфейсів. Методи, які замінюють віртуальні методи в батьківському класі, потребують ключового слова override, щоб запобігти випадковому перевизначенню. У Visual C# структура подібна до полегшеного класу: це стекований тип, який реалізує інтерфейси, але не підтримує успадкування.

На додаток до основних об'єктно-орієнтованих принципів, описаних вище, Visual C# спрощує розробку програмних компонентів за допомогою кількох інноваційних мовних конструкцій, зокрема таких:

- Сигнатури інкапсульованих методів, які називаються делегатами, які підтримують жорстко введені повідомлення про події.

- Властивості, які функціонують як об’єкти доступу для закритих змінних-членів.

- Атрибути з декларативними метаданими про типи середовища виконання.

- Вбудовані коментарі до документації XML.

- LINQ (Language-Integrated Query), який надає вбудовані можливості запитів до кількох джерел даних.

Якщо потрібно взаємодіяти з іншим програмним забезпеченням Windows, таким як об’єкти COM або рідні бібліотеки DLL Win32, є можливим використовувати процес під назвою «Interop» у Visual C#. Процес Interop дозволяє програмам Visual C# робити майже все, що може робити рідна програма C++. Visual C# навіть підтримує покажчики та концепцію «небезпечного» коду, коли прямий доступ до пам’яті критичний.

Порівняно з C і C++ процес збирання Visual C# простіший і гнучкіший, ніж Java. Не існує окремих файлів заголовків, а методи та типи не потрібно оголошувати в певному порядку. У вихідному файлі Visual C# можна визначити будь-яку кількість класів, структур, інтерфейсів і подій. Після врахування всього було обрано Visual C# для програмної реалізації системи резервного копіювання даних у хмарі, віртуальному та фізичному середовищі.

2.3 Розгорнута постановка завдання

Відповідно до технічного завдання на випускну кваліфікаційну роботу другого магістерського рівня необхідно впровадити програмне забезпечення для хмарної, віртуальної та фізичної системи резервного копіювання даних та провести відповідні дослідження. Відповідно до цілей резервного копіювання визначаються такі завдання резервного копіювання:

- Відбір цільових даних.

- Зберігання зазначених даних для подальшого відновлення.

- Відновлення збережених даних.

- Забезпечення стабільності збережених даних від зміни та знищення.

- Обмеження доступу до збережених даних.

- Забезпечення контролю над системою та процесом резервного копіювання. У процесі випускної кваліфікаційної роботи необхідно виконати наступний обсяг робіт:

а) Провести аналіз і дослідження існуючих аналогових систем для виявлення їх позитивних і негативних характеристик. Результати аналізу будуть враховані при подальшій розробці;

б) обрати та обґрунтувати спосіб побудови системи резервного копіювання даних хмарного, віртуального та фізичного середовищ. Розробити функціональні та структурні схеми системи;

в) розробити програмне забезпечення системи, яке дозволить реалізувати завдання, визначене технічним завданням. Будувати структурні схеми програмування алгоритмів і підпрограм;

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

д) розробити рекомендації щодо організаційно-методичних заходів щодо введення системи в промислову експлуатацію та її подальшої успішної експлуатації;

е) зробити висновки щодо обсягу виконаної роботи та досягнутих результатів.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Для створення надійної системи необхідний набір адміністративних і технічних заходів для встановлення пріоритетів інформаційних активів, аналізу середовища, планування графіка резервування, розробки нормативних процедур і документів [10].

Існує два основних способи зберігання даних - резервне копіювання зображень і резервне копіювання файлів [11]. Обидва методи дозволяють отримувати певні файли та дані. Однак резервне копіювання зображень є розширеною версією, оскільки передбачає роботу з системними метаданими та надає додаткові параметри відновлення.

Резервне копіювання файлів є найстарішим і все ще популярним способом захисту інформації від втрати. Усі файли та папки копіюються на резервні носії, що дуже схоже на звичайну практику перезапису особистих файлів на зовнішньому диску або в іншій папці на комп’ютері.

Резервне копіювання образів дозволяє уникнути більшої частини навантаження на систему, яка неминуча під час пошуку файлів. Повна копія диска створюється простим копіюванням блоків по одному. Крім того, програма розпізнає та копіює лише ті блоки, які були змінені з часу останньої копії. Таким чином, якщо у файлі розміром 2 ГБ відбудеться невелика зміна, під час резервного копіювання файлів буде перезаписано всі 2 ГБ, а під час резервного копіювання образу буде перезаписано лише змінений блок. Цей алгоритм дозволяє дуже швидко створювати резервні копії.

Технологія резервного копіювання образів зазвичай дозволяє переглядати копію як додатковий диск. Адміністратор може переглядати його вміст і при необхідності відновлювати окремі файли. Резервне копіювання образів і файлів

може бути повним, диференціальним або додатковим. Усі три типи дозволяють виключати певні типи даних із копіювання, якщо вони з якихось причин не потрібні (наприклад, тимчасові файли).

Інтервал часу, після якого необхідно створити резервну копію даних, визначається цільовою точкою відновлення (RPO- Recovery Point Objective) директиви. Він може відрізнятися для різних систем і навантажень: п’ять хвилин, година або цілий день. Якщо РРО становить 30 хвилин, точку відновлення потрібно створювати кожні півгодини. Оскільки створення резервної копії вимагає великої кількості обчислювальних ресурсів, робота інших програм повинна на деякий час припинитися. На кілька годин або кілька секунд - це залежить від плану і методу резервного копіювання, потужності системи і навіть характеру бізнесу. Період часу, протягом якого система може бути призупинена для резервного копіювання, називається вікном резервного копіювання.

Програми резервного копіювання можуть отримати доступ до системних даних двома способами:

— За допомогою агента. У цьому випадку на кожну фізичну та віртуальну машину (VM) встановлюється невелика програма-агент.

— Без агента. У хмарних і віртуальних середовищах кількість віртуальних машин може бути досить великою, тому зручніше використовувати безагентне резервне копіювання. У цій версії також використовуються агенти, але в дуже невеликій кількості, що означає, що процесом копіювання легше керувати. Агент зазвичай встановлюється на кожному віртуальному хості, який також знаходиться на віртуальній машині. Агент обмінюється даними з хостом і копіює всі віртуальні машини, які знаходяться на ньому. Більшість систем мають кілька хостів, і віртуальні машини можуть переміщатися між ними, тому системи резервного копіювання повинні постійно контролювати їхній стан.

Звичайний процес резервного копіювання ділиться на такі основні частини [12]:

-Періодичний запуск копіювання.

-Запуск відновлення на вимогу.

-Тестування процесу копіювання. Дискові масиви можна використовувати для надійного зберігання даних, щоб уникнути втрати даних у разі виходу з ладу одного з дисків.

Накопичувачі на основі використання енергонезалежної пам'яті (NAND SSD) з'явилися порівняно недавно, але завдяки значно нижчій ціні (від 1 долара США за гігабайт) почали впевнено завойовувати ринок. До недавнього часу вони значно поступалися традиційним накопичувачам - жорстким дискам - у швидкості запису, але компенсували це високою швидкістю пошуку інформації (початкове позиціонування). З 2012 року SSD виробляються зі швидкістю читання та запису в рази вищою, ніж жорсткі диски. Вони характеризуються відносно невеликими розмірами і низьким енергоспоживанням. RAM SSDS. Ці накопичувачі засновані на використанні енергонезалежної пам’яті (та сама пам’ять, яка використовується в оперативній пам’яті персонального комп’ютера) як накопичувач RAM і характеризуються надшвидким читанням, записом і отриманням інформації. Їх основним недоліком є надзвичайно висока вартість. Вони в основному використовуються для прискорення роботи великих систем баз даних і потужних графічних робочих станцій. Як правило, такі накопичувачі оснащені батареями для збереження даних у разі відключення електроенергії, а дорожчі моделі оснащені системами резервного та/або оперативного копіювання. Прикладом таких накопичувачів є I-RAM. Користувачі з достатньою кількістю оперативної пам’яті можуть організувати віртуальну машину та розмістити її жорсткий диск у оперативній пам’яті для оцінки продуктивності.

Переваги

– Відсутність частин, що рухаються.

– Повна відсутність шуму (0 дб).

– Висока механічна стійкість (короткочасно витримують порядку 1500 g).

– Стабільність часу зчитування файлів поза залежністю від їхнього

розташування або фрагментації; більше того, блоки, що йдуть підряд з погляду операційної системи, через вирівнювання зношування (wear leveling) будуть розташовані у випадковому порядку.

– Швидкість читання/запису вище, ніж у розповсюджених жорстких дисків, і близька до пропускної здатності інтерфейсів (SAS/SATA II 300 Мбайт/с, SAS/SATA III 600 Мбайт/с). Для твердотільних накопичувачів були розроблені більше швидкі інтерфейси: mSATA, NGFF (M.2), SATA Express, NVM Express (стандарт на підключення SSD по шинах PCI Express).

– Кількість випадкових операцій уведення-виводу в секунду (IOPS) в SSD на кілька порядків вище, ніж у жорстких дисків.

– Низьке енергоспоживання.

– Широкий діапазон робочих температур.

– Набагато менша чутливість до зовнішніх електромагнітних полів.

– Малі габарити й вага.

Недоліки

Основним недоліком NAND SSD є обмежена кількість циклів перезапису. Звичайна флеш-пам'ять (MLC, багаторівнева комірка, багаторівневі комірки пам'яті) дозволяє записувати дані приблизно 3000-10000 разів. Більш дорогі види пам'яті (SLC, Single-level cell, однорівневі осередки пам'яті) - приблизно в 100 000 разів. Для боротьби з нерівномірним зносом використовуються схеми балансування навантаження. Контролер зберігає інформацію про те, скільки разів які блоки були перезаписані, і при необхідності «замінює» їх. Коли ресурс створюється, диск переходить у режим лише для читання, щоб дозволити копіювання даних. Цього недоліку немає в RAM SSD, а також у кількох перспективних технологіях, які можуть замінити флеш-пам’ять до кінця 2010 року, наприклад FRAM, де ресурс може перебувати в режимі безперервного перезапису протягом десятиліть.

Ціна гігабайта SSD в кілька разів (у 6-7 для найдешевшої флеш-пам'яті) перевищує ціну гігабайта HDD (на жовтень 2014 року - 35 центів за гігабайт).

Причому ціна SSD прямо пропорційна їх ємності, тоді як ціна традиційних HDD залежить не тільки від кількості пластин, але і від кількості пластин, і вони зростають повільніше зі збільшенням обсягу накопичувача. Використання команди TRIM на SSD може ускладнити або зробити неможливим відновлення видаленої інформації за допомогою засобів відновлення. Неможливість відновлення інформації при падінні напруги. Оскільки контролер і носій даних у SSD знаходяться на одній платі, перевищення або падіння напруги найчастіше призводить до згоряння всього носія SSD із незворотною втратою інформації. І навпаки, з жорсткими дисками частіше згорає лише плата контролера, що дозволяє відновити інформацію з прийнятною працею.

У Windows 7 введена спеціальна оптимізація для роботи з SSD. Наприклад, Windows 7 не застосовує дефрагментацію, технології Superfetch і ReadyBoost, а також інші методи читання до SSD, що не дозволяє додаткам швидше завантажуватися зі звичайних жорстких дисків. Попередні версії Microsoft Windows такої спеціальної оптимізації не мають і призначені для роботи тільки зі звичайними жорсткими дисками. Тому, наприклад, деякі операції з файлами в Windows Vista, які не вимкнено, можуть скоротити термін служби SSD. Операцію дефрагментації слід вимкнути, оскільки вона практично не впливає на продуктивність SSD і лише додатково зношує ресурс SSD. Комп’ютери Mac OS X і Macintosh з SSD. Починаючи з версії 10.7 (Lion), Mac OS X повністю підтримує TRIM для твердотільної пам’яті, встановленої в системі. З 2010 року Apple випускає серію комп’ютерів Air, оснащених твердотільною пам’яттю на основі флеш-пам’яті NAND. До 2010 року для цього комп'ютера покупець міг вибрати стандартний жорсткий диск, але подальший розвиток лінійки на користь максимальної легкості і менших розмірів корпусів комп'ютерів цієї серії зумовили повну відмову від стандартних жорстких дисків на користь твердотільних накопичувачів.

Обсяг пам'яті, включеної в комп'ютери серії Air, коливається від 128 ГБ до 512 ГБ. За даними Дж. П. Моргана, з моменту подання було продано 420 000 комп’ютерів цієї серії повністю напівпровідникових флеш-комп’ютерів NAND.

11 червня 2012 року була представлена оновлена лінійка професійних ноутбуків MacBook Pro з дисплеєм Retina на основі флеш-пам'яті, яка включала опціональну флеш-пам'ять на 768 ГБ

GNU/Linux і комп’ютери на цій платформі з SSD. Починаючи з версії ядра 2.6.33, операційна система Linux повністю підтримує TRIM для SSD-накопичувачів, установлених у системі, якщо в налаштуваннях монтування диска вказано параметр «відкинути».

Перспективи розвитку

Основний недолік SSD на основі флеш-пам'яті - обмежена кількість циклів перезапису - можна усунути з розвитком технологій виробництва енергонезалежної пам'яті шляхом виготовлення носія інформації з інших фізичних принципів, таких як FeRam, ReRAM (резистивна оперативна пам'ять), тощо.

Твердотільні накопичувачі (SSD) характеризуються високою швидкістю читання, що робить їх ідеальними як системні накопичувачі.

У підсумку виявляється, що недоліків вистачає. Основна перевага полягає в тому, що RAID-масив, створений на правильному контролері, може фактично збільшити швидкість дискових операцій. Масив RAID на основі SSD може бути надшвидким для читання та досить посереднім для запису.

3.2 Розробка структурної схеми

Для розробки структурної схеми системи необхідно розглянути рівні RAID-масивів.

Розрізняють кілька основних рівнів RAID-масивів[13][14]: RAID 0, 1, 2, 3, 4, 5, 6, 7. Також існують комбіновані рівні, такі як RAID 10, 0+1, 30, 50, 53 і т.п. Розглянемо принципи функціонування, переваги й недоліки основних рівнів.

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

Дисковий масив без надлишкового зберігання даних. Інформація розбивається на блоки, які одночасно записуються на окремі диски, що забезпечує підвищення продуктивності. Такий спосіб зберігання інформації ненадійний, оскільки вихід з строю одного диска приводить до втрати всієї інформації, тому рівнем RAID [15] як таким не є.

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

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

Дисковий масив з дублюванням інформації (віддзеркалюванням даних). У найпростішому випадку два накопичувачі містять однакову інформацію і є одним логічним диском. При виході з ладу одного диска його функції виконує інший. Для реалізації масиву потрібно не менше двох вінчестерів. RAID 1 – найпростіший відказостійкий масив.

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

Схема резервування даних з використанням коду Хеммінга (Hamming code) для корекції помилок. Потік даних розбивається на слова – причому розмір слова відповідає кількості дисків для запису даних. Для кожного слова

обчислюється код корекції помилок, що записується на диски, виділені для зберігання контрольної інформації. Їхнє число дорівнює кількості біт у слові контрольної суми.

RAID 2 не одержав комерційного застосування.

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

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

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

Цей рівень має набагато меншу надлишковість, ніж RAID 2. У RAID 2 більшість дисків, які зберігають керуючу інформацію, необхідні для визначення несправного біта. Як правило, контролери RAID можуть отримувати дані про помилки за допомогою механізмів відстеження випадкових збоїв. Завдяки розділенню даних RAID 3 має високу продуктивність. Оскільки кожна операція вводу/виводу передається майже на всі диски в масиві, одночасна обробка кількох запитів неможлива. Цей рівень підходить для додатків з великими файлами та низьким оборотом (переважно мультимедіа). Використання тільки одного диска для зберігання керуючої інформації пояснює той факт, що коефіцієнт використання дискового простору досить високий (отже - відносно

низька вартість). Для реалізації масиву вам потрібно принаймні три жорсткі диски.

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

Цей масив дуже схожий на рівень RAID 3. Потік даних ділиться не на рівні байтів, а на рівні блоків інформації, кожен з яких записується на окремий диск.

Після запису групи блоків обчислюється контрольна сума та записується на спеціальний диск. Кілька операцій читання можна виконувати одночасно в RAID 4. Цей масив покращує продуктивність передачі невеликих файлів (шляхом розпаралелювання операції читання). Але оскільки контрольна сума на виділеному диску повинна змінюватися під час операції запису, то одночасне виконання операцій неможливо (при наявності асиметрії операцій введення і виведення).

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

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

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

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

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

RAID 6 — це відказостійкий масив незалежних дисків з розподілом контрольних сум, обчисленими двома незалежними способами. Цей рівень багато в чому схожий на RAID 5, за винятком того, що він використовує не одну, а дві незалежні схеми контролю парності, що дозволяє системі залишатися працездатною, якщо два диски виходять з ладу одночасно. Для обчислення контрольних сум у RAID 6 використовується алгоритм Ріда-Соломона (Reed-Solomon).

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

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

RAID 10

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

Крім базових рівнів RAID 0 – RAID 5, описаних у стандарті, існують комбіновані рівні RAID 1+0, RAID 3+0, RAID 5+0, RAID 1+5, які різні виробники інтерпретують кожний на свій лад.

Matrix RAID

Matrix RAID – це технологія, реалізована фірмою Intel у своїх чипсетах починаючи з ICH6R. Жорстко кажучи, ця технологія не є новим рівнем RAID (її аналог існує в апаратних RAID-контролерах високого рівня), вона дозволяє, використовуючи невелику кількість дисків організувати одночасно один або

кілька масивів рівня RAID 1, RAID 0 і RAID 5. Це дозволяє за порівняно невеликі гроші забезпечити для одних даних підвищену надійність, а для інших високу швидкість доступу.

Програмний RAID

Для реалізації RAID можна використовувати не тільки апаратні, але й повні програмні компоненти (драйвери). Наприклад, у системах на основі ядра Linux існують спеціальні модулі ядра, і ви можете керувати пристроями RAID у GNU/Linux за допомогою інструменту mdadm.

Розроблена в результаті роботи система підвищення надійності зберігання інформації на SSD за технологією RAID заснована на використанні технології Intel® SATA Solid-State Drive, з використанням RAID 0 і RAID 1 або RAID 6, залежно від кількості дисків у RAID-масиві. Якщо дисків два, то використовується RAID 1, якщо більше одного, то використовується RAID 6.

Intel також анонсувала новий, більш гнучкий дисковий контролер під назвою Intel® SATA Solid-State Drive. Він може підтримувати кілька одночасних масивів RAID на одному наборі дисків. Масив RAID використовує кілька дисків одночасно, що дозволяє підвищити продуктивність підсистеми зберігання даних, одночасно забезпечуючи захист від збою диска.

Структурна схема розробленої системи наведена на рисунку 3.1.

Драйвер твердотільного накопичувача Intel® SATA підтримує різні режими міграції, включаючи перехід від одного диска до масиву RAID або додавання диска до існуючого масиву RAID. Контролер SATA підтримує «власну» чергу команд, а також швидкість 300 МБ/с для кожного порту SATA. Однак ця швидкість навряд чи дасть якусь перевагу, оскільки швидкість читання з плат не перевищує 80 МБ/с, а швидкість читання з кеша жорсткого диска мало впливає на продуктивність.

Розподілення на зони збереження системи резервного копіювання даних хмарного, віртуального й фізичного середовища

Рисунок 3.1 – Структурна схема системи

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

Реально програмне забезпечення створює віртуальні диски на яких

імітується RAID-масив. Для цього створюється як мінімум 2 віртуальні диски, які дозволяють гарантовано зберігати інформацію за допомогою RAID 0 та RAID 1, якщо дисків створюється більше ніж 2 то гарантовано зберігається інформація за допомогою RAID 0 та RAID 6.

3.3 Розробка функціональної схеми

Для розробки функціональної схеми системи необхідно визначити базові функції системи.

Базові функції дозволяють працювати за графіком, стискати і шифрувати інформацію, що копіюється, а додаткові – відрізняються великою різноманітністю. Наприклад:

- Дублювання. Збереження «дублікатів» досягається шляхом копіювання на різні джерела.
- Дедуплікація. Аналізує і стискає файли, зменшуючи навантаження на канали передачі і файлосховище.
- Образ системи. Піднімає працездатність при ушкодженнях ОС або ПК.
- Оптимізація навантаження. Швидкість виконання бекапа.
- Сумісність з різними ОС і СКБД. Копії файлів і БД створюються з урахуванням динаміки змін структури в момент копіювання.
- Інструментарій для автоматизації віддаленої роботи адміністраторів.
- Підтримка віртуальних пристроїв.
- Підтримка хмарних сховищ.
- Алгоритми відновлення, розраховані на різні швидкості.

На рисунку 3.2 зображена функціональна схема системи.

Програмне забезпечення системи складається з наступних блоків:

– Блок розподілу інформації за критерієм критичності/важливості визначає який саме вид RAID-масиву необхідно буде використовувати. Якщо інформація не є критичною то буде використовуватися RAID 0.

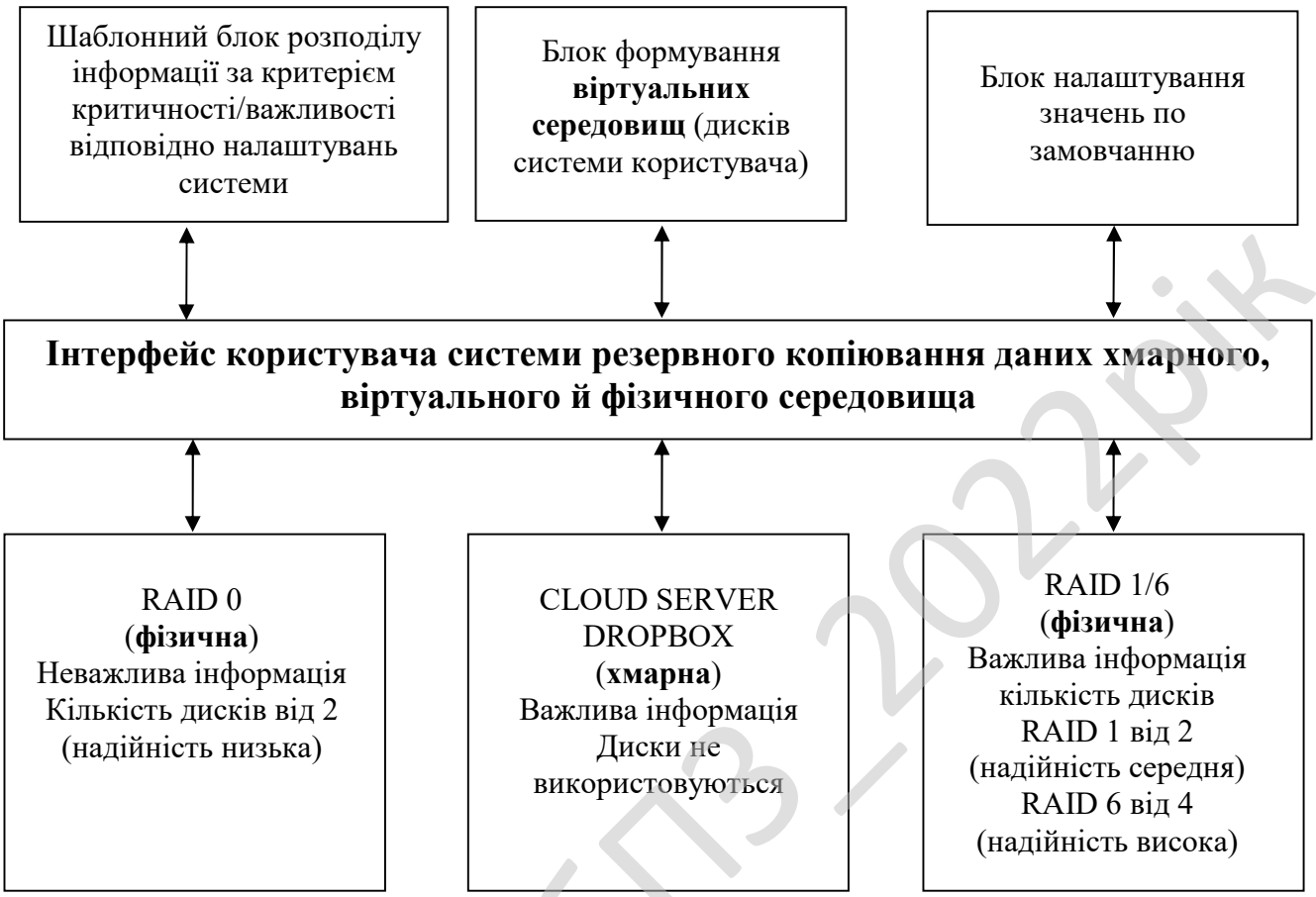

Рисунок 3.2 – Функціональна схема системи

Якщо інформація є важливою, то в залежності від ступеня важливості, та вимог до швидкодії, буде використовуватися або RAID 1, який є більш швидким, та менш надійним, або RAID 6, який є дуже надійним але менш швидким.

Блок вибору кількості дисків – призначений для визначення кількості дисків з яких буде складатися RAID-масив. Якщо дисків буде 2 то буде використовуватися технологія RAID 0 та RAID 1. Якщо дисків буде більше 2 то буде використовуватися технологія RAID 0 та RAID 6.

Блок формування віртуальних дисків використовується у разі коли диск тільки один. Тоді на цьому диску формуються декілька віртуальних дисків, з якими відбуваються дії аналогічні як ті, що відбуваються над фізичними дисками. Тобто RAID-масив формується з віртуальних дисків.

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

У блоці RAID 6 використовується кодек Ріда-Соломона.

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

3.4 Розробка діаграми процесів

Розглянемо розроблену діаграму процесів системи резервного копіювання даних хмарного, віртуального й фізичного середовища, яка зображена на рисунку 3.3. Основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей.

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.

Після початку роботи розробленого програмного забезпечення ми потрапляємо до головного блоку системи звідки через ланку дій відбувається наступне:

– Інтерфейс ПЗ системи резервного копіювання.

– Вибір конфігурації системи.

– Надмірність кодування.

– Загальна кількість дисків.

– Кодування SSD диску.

– Створення RAID масиву.

– Шифруючий фільтр.

– Встановлення паролю на закодовані дані.

– Декодування SSD диску.

– Відновлення пошкоджених даних.

– Перевірка цілісності даних.

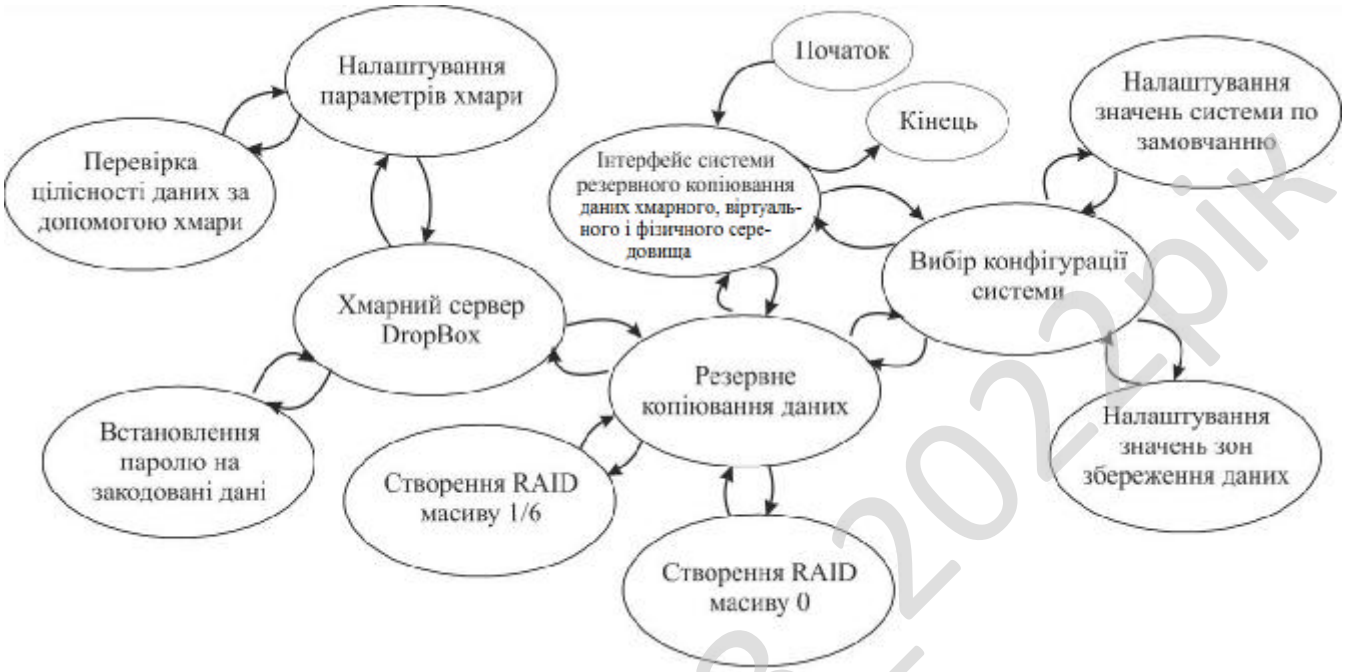

Рисунок 3.3 – Діаграма взаємодії процесів

Використовується проектна модель, графічне представлення «потоків» даних у системі. Вважається звичайною практикою, коли розробник спочатку малює діаграму взаємодії процесу даних на рівні контексту, щоб показати, як взаємодіє система. Ця діаграма додатково вдосконалена з детальним описом процесів і потоків даних, щоб показати систему, що розробляється. Розглянувши опис системи, схему структур, функціональну схему системи та діаграму взаємодії процесів, переходимо до опису блок-схем алгоритму, що було написано для реалізації системи.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Для реалізації програмного забезпечення системи резервного копіювання даних хмарного, віртуального й фізичного середовища були проведені розрахунки й підібрані набори тестових даних для перевірки правильності реалізації проектних рішень.

Було створено блок-схеми роботи системи. Перед їх розглядом необхідно провести роз’яснення, який саме тип блок-схем використовується. Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму. В інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів. Функціональні схеми можуть виконуватися в укрупненому й розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними. У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення,

зумовлений процес (підпрограма), дані та з'єднувач. Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія. Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції. Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу.

Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи. Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані. Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи). З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Блок-схеми показують весь процес роботи системи з підсистемами та частково доказують правильність вибраних проектних рішень. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано,

що виходячи з вибору мови програмування й інших технічних засобів, програма буде об’єктно-орієнтована, що вимагає високого рівня декомпозиції задач на класи.

Алгоритм, що пропонується дозволяє комплексно проаналізувати інфраструктуру, виділити основні вимоги до системи, можливі обмеження, що накладаються середовищем, і ефективно впровадити розроблене рішення. Проектування системи резервного копіювання можна умовно розділити на два етапи, що складаються з декількох кроків.

Етап 1 – адміністративний:

1. Визначення обсягу даних, що копіюються. Найважливішим кроком можна вважати визначення найбільш критичної інформації, втрата якої може негативно або згубно позначитися на бізнес-процесах. Правильно провівши межу між ключовими та допоміжними даними, можна істотно скоротити витрати на систему резервного копіювання, зменшивши час зняття копій і необхідний обсяг дискового простору.

2. Завдання частоти зняття копій. На цьому кроці необхідно визначити мінімальний відрізок часу, втрата даних за який неприйнятно. Даний параметр цілком залежить від частоти зміни позначених даних. Від нього залежить не тільки обсяг сховища резервних копій, а й версійність даних. Для кожного інформаційного активу цей параметр повинен бути заданий індивідуально.

3. Оцінка зміни даних. Цей параметр висловлює кількісну зміну даних між операціями резервного копіювання. Даний відсоток визначається експериментально і служить основою для довгострокового планування обсягу носіїв і ємності інтерфейсів системи резервного копіювання.

4. Визначення допустимого часу зняття резервних копій. Планування «вікна» резервного копіювання, коли дані не використовуються або використовуються тільки для читання. Це той час, протягом якого можна проводити резервне копіювання. Найчастіше воно потрапляє на неробочий час – вечір і ніч. У тому випадку, коли дані повинні бути доступні для читання і запису

постійно (системи високої доступності), слід використовувати спеціальні засоби для копіювання, такі як VSS, снепшот файлової системи.

5. Визначення необхідної ємності для системи резервного копіювання. Під ємністю розуміється кількість даних, які система може прийняти під час вікна копіювання.

6. Визначення правил зберігання резервних копій. На цьому кроці слід встановити місце зберігання оперативної копії (це може бути система з низьким часом доступу і високою пропускною здатністю, що знаходиться в безпосередній близькості від систем-клієнтів), місце зберігання старіших, але актуальних копій: порядок приміщення копії в архів, ступінь дедуплікації , а також зберігання копій відповідно до політики інформаційної безпеки підприємства та допуску співробітників до сховищ копій.

7. Визначення методу перевірки резервних копій. Якщо під час зняття копій сталася помилка, вона може залишитися непоміченою аж до спроби відновлення з цієї копії. Перевірка найчастіше здійснюється математичними методами самого програмного забезпечення системи резервного копіювання, але найбільш надійним є метод відновлення даних «в пісочниці» для перевірки їх цілісності. Цей крок покликаний регламентувати методи захисту від помилок.

8. Визначення методу утилізації носіїв. Носії з вичерпаним терміном придатності можуть бути поміщені в архів, а можуть бути утилізовані. На цьому кроці слід вказати метод знищення неактуальних резервних копій. Він повинен бути узгоджений з політикою інформаційної безпеки підприємства.

Етап 2 – проектування (аналіз наявної) системи резервного копіювання:

1. Визначення пропускної здатності інтерфейсу системи резервного копіювання. Необхідна швидкість отримання даних для завершення створення копії, не виходячи за рамки вікна резервного копіювання, обчислюється як відношення обсягу копійованих даних D до вікна резервного копіювання w.

2. Визначення пропускної здатності інтерфейсу клієнта системи, тобто швидкості, з якою клієнт системи резервного копіювання може приймати і

отримувати від неї дані. Дорівнює найменшою з швидкостей читання / запису накопичувача або каналу передачі інформації.

3. Вибір оптимального алгоритму зняття резервних копій. Від вибору алгоритму резервного копіювання залежить дуже багато чого, зокрема час копіювання та відновлення, об'єм переданих даних, обсяг сховища резервних копій, метод їх зберігання.

4. Визначення часу зняття й відновлення копій (включає в себе час читання і запису, пропускну здатність каналу). На цьому кроці визначається реальна швидкість копіювання й відновлення даних.

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 підсистеми та останньої стадії – перевірка поточного стану з завершенням роботи системи резервного копіювання. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Основну частину функціональності файлів у мережевих сховищах забезпечують файлові системи та бази даних; їх доповнюють додатки управління зберіганням, наприклад, операції резервного копіювання, що також є файловими додатками.

Розглянемо опис алгоритмів функціонування системи. Нижче наведемо ту частину коду, яка виконує вищеперераховані дії. Використанний кодер Ріда-Соломона. Дані, що кодуються, передаються через масив data[i], де i = 0...(k – 1), а згенеровані символи парності заносяться в масив b[0]...b[2t – 1]. Вихідні й результуючі дані повинні бути представлені в поліноміальній формі (тобто у звичайній формі машинного подання даних).

Кодування виробляється з використанням зсувного feedback-регістра,

заповненого відповідними елементами масиву g[] з породженим поліномом усередині. Згенероване кодове слово описується наступною формулою:

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

де ^ означає зведення в ступінь.

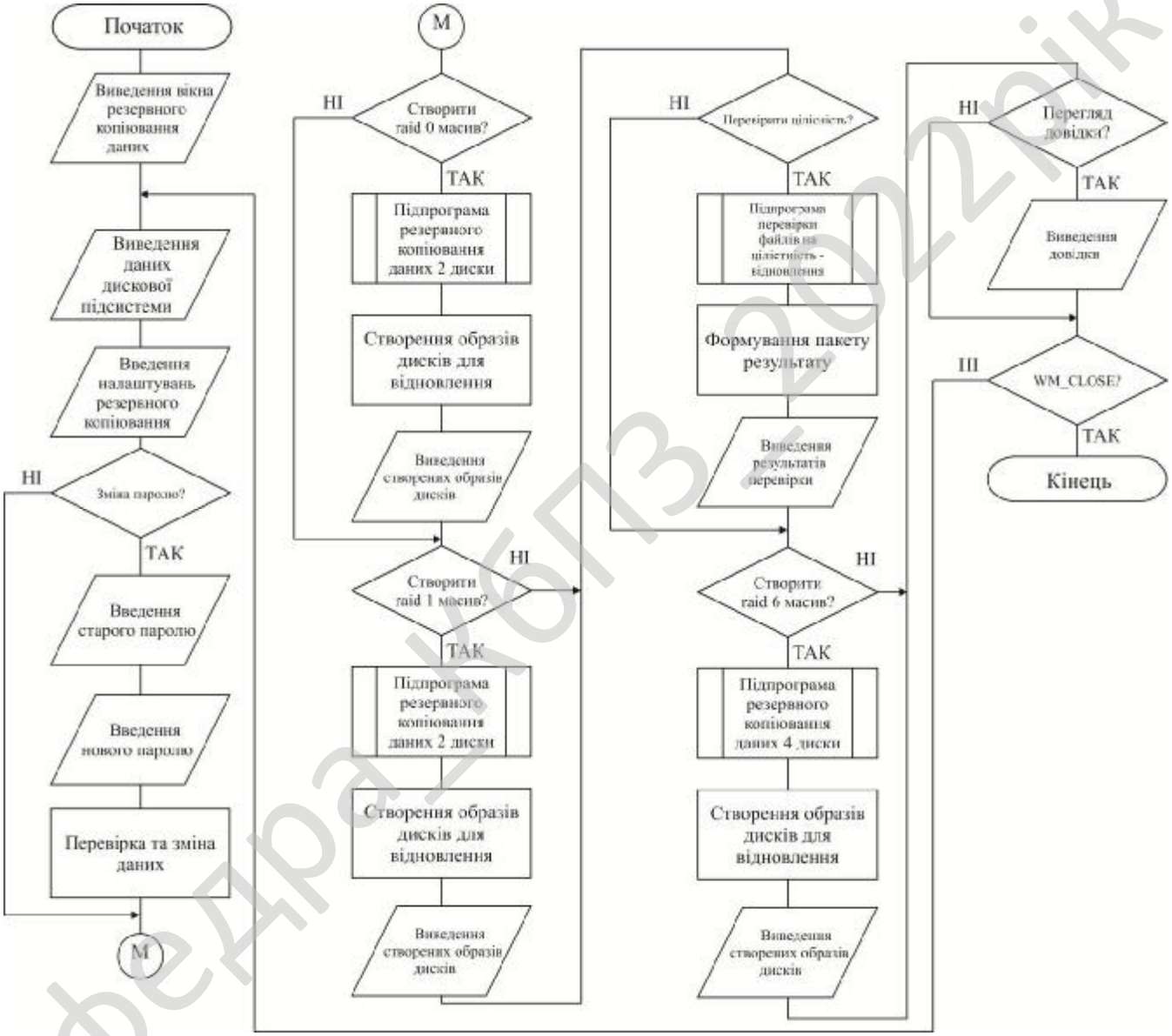

Рисунок 4.1 – Блок-схема основної програми

```
encode_rs()
{
        int i, j;
        int feedback;
```

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

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

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

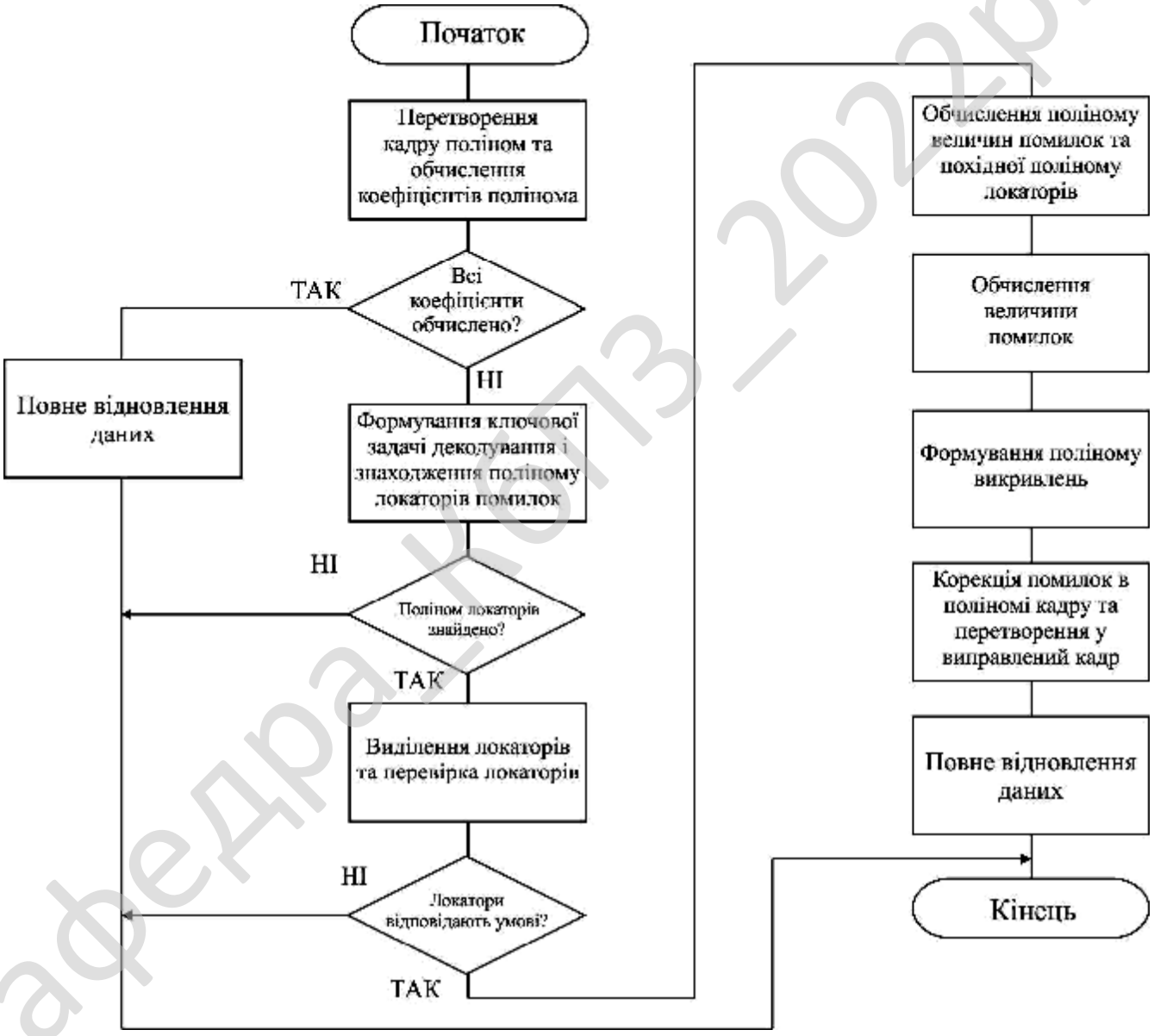

Рисунок 4.2 – Блок-схема роботи підпрограми

Типова схема декодування, що одержала назву авторегрессійного спектрального методу декодування, складається з наступних кроків:

– Обчислення синдрому помилки (синдромний декодер).

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

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

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

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

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

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

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

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

всіх інших випадках відновлення оригінального вмісту перекручених символів цілком можливо.

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

Система керування базами даних (СКБД, Database Management System, DBMS) – набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних.

У реляційних СКБД використовувалися непроцедурні мови керування даними (SQL) і передбачався значний ступінь незалежності даних. Реляційні системи внесли значні удосконалення в управління даними такі як: графічний користувацький інтерфейс (GUI), клієнт-серверні застосунки, розподілені бази даних, паралельний пошук даних та інтелектуальний аналіз даних. Відповіддю на зростаючу складність програм баз даних стали два нових напрямки розвитку СКБД: об'єктно-орієнтовані СКБД і об'єктно-реляційні СКБД (ООСКБД).

СКБД третього покоління повинні бути відкриті для інших підсистем. Це включає оснащення різноманітними інструментами підтримки прийняття рішень, доступом з багатьох мов програмування, інтерфейсами до існуючих популярних систем і бізнес-застосунків, можливістю запуску програм з бази даних на іншій машині і розподілені СКБД. Весь набір інструментів і СКБД має ефективно функціонувати на різноманітних апаратних платформах з різними операційними системами.

Крім того, СКБД, що розраховує на широку сферу застосування, повинна бути оснащена мовою четвертого покоління (4GL).

Розвиток індустрії систем керування базами даних базується на значних фундаментальних наукових дослідженнях. В даний час дослідники мають у своєму розпорядженні засоби, що дозволяють ефективно реалізувати найскладніші запити, що маніпулюють терабайтами й петабайтами різних даних.

Основними тенденціями, які дали привід для проведення різних масштабних досліджень в області баз даних стали:

– Експонентний ріст даних.

– Значне ускладнення структур використовуваних даних.

– Широке поширення дешевих високопродуктивних апаратних засобів.

– Активний розвиток засобів комунікації та «всесвітньої павутини» World Wide Web.

– Поява нових важливих областей застосування СКБД. У першу чергу, це пов'язано з інтелектуальним аналізом даних, сховищами даних, а останнім часом – з паралельними обчисленнями і хмарними технологіями.

Основні характеристики СКБД

1. Контроль за надлишковістю даних.

2. Несуперечливість даних.

3. Підтримка цілісності бази даних (коректність та несуперечливість).

4. Цілісність описується за допомогою обмежень.

5. Незалежність прикладних програм від даних.

6. Спільне використання даних.

7. Підвищений рівень безпеки.

Можливості СКБД

1. Дозволяється створювати БД [29] (здійснюється за допомогою мови визначення даних DDL (Data Definition Language)).

2. Дозволяється додавання, оновлення, видалення та читання інформації з БД (за допомогою мови маніпулювання даними DML, яку часто називають мовою запитів).

3. Можна надавати контрольований доступ до БД за допомогою: Системи забезпечення захисту, яка запобігає несанкціонованому доступу до БД; Системи керування паралельною роботою прикладних програм, яка контролює процеси спільного доступу до БД; Система відновлення – дозволяє відновлювати БД до

попереднього несуперечливого стану, що був порушений в результаті збою апаратного або програмного забезпечення.

Основні компоненти середовища СКБД

1. Апаратне забезпечення.

2. Програмне забезпечення.

3. Дані.

4. Процедури – інструкції та правила, які повинні враховуватись при проектуванні та використанні БД.

5. Користувачі: адміністратори даних (керування даними, проектування БД, розробка алгоритмів, процедур) та БД (фізичне проектування, відповідальність за безпеку та цілісність даних); розробники БД; прикладні програмісти; кінцеві користувачі.

Архітектура СКБД

Існує трирівнева система організації СКБД ANSI-SPARC, при якій існує незалежний рівень для ізоляції програми від особливостей представлення даних на нижчому рівні.

Рівні:

1. Зовнішній – представлення БД з точки зору користувача.

2. Концептуальний – узагальнене представлення БД, описує які дані зберігаються в БД і зв'язки між ними. Підтримує зовнішні представлення, підтримується внутрішнім рівнем.

3. Внутрішній – фізичне представлення БД в комп'ютері.

Логічна незалежність – повна захищеність зовнішніх моделей від змін, що вносяться в концептуальну модель.

Фізична незалежність – захищеність концептуальної моделі від змін, які вносяться у внутрішню модель.

Firebird (також Firebird SQL) – компактна, крос-платформова, вільна реляційна система керування базами даних, що реалізує більшість функцій

стандарту SQL 2003. Вона може запускатись на більшості UNIX-подібних систем (в тому числі Linux та FreeBSD) та Windows.

Основні можливості

Відповідність вимогам ACID: Firebird спеціально спроектовано таким чином, щоб задовільняти вимоги «атомарності, несуперечності, ізоляції та довговічності» транзакцій «Atomicity, Consistency, Isolation and Durability».

Версійна архітектура: основна особливість Firebird – версійна архітектура, що дозволяє серверу обробляти різні версії одного запису в будь-який час таким чином, що кожна транзакція бачить свою версію даних, не заважаючи сусіднім. Таким чином, транзакції, що читають, не блокують транзакції, що пишуть і навпаки. Окрім того, це дає можливість відмовитись від логу транзакцій і таким чином зменшити ймовірність пошкодження службової інформації бази даних.

Збережені процедури: за допомогою мови PSQL (процедурна SQL) можна створювати складні збережені процедури для обробки даних на боці сервера. Таким чином можна виносити на сторону сервера значну частину бізнес-логіки програмного пакету чи формувати дані для звітів.

Події: збережені процедури та тригери можуть ґенерувати події, на які, в свою чергу, може підписатися клієнтська програма і відповідним чином їх обробляти.

Генератори: дають можливість просто реалізовувати автоінкрементні поля; оскільки вони працюють незалежно від транзакцій, то можуть використовуватись для генерації первинних ключів чи керування тривалими запитами в інших транзакціях.

Повний контроль над транзакціями: одна клієнтська програма може одночасно виконувати декілька транзакцій, включно з різними рівнями ізоляції. Окрім того, доступні протокол двофазного підтвердження транзакцій (що забезпечує гарантовану стійкість при роботі з кількома БД), оптимістичне блокування даних (блокується не вся сторінка даних, а лише змінені записи), точки збереження транзакцій та автономні транзакції (починаючи з версії 2.5).

Резервне копіювання на льоту: завдяки в тому числі й версійній архітектурі немає потреби зупиняти сервер для резервування бази даних. Процес резервного копіювання зберігає стан бази даних на момент початку резервування, не шкодячи роботі інших клієнтів. Додатково існує можливість інкрементного резервування бази даних.

Тригери: для будь-якої таблиці можна назначити декілька тригерів, що спрацюють до чи після додавання, оновлення чи вилучення даних. У тригерах використовується мова PSQL, що дозволяє задавати початкові значення даних, перевіряти їх цілісність, збуджувати події та ін. Починаючи з версії 1.5 у Firebird з'явились універсальні тригери, що дозволяють обробляти вставку, оновлення та вилучення даних в одному місці.

Зовнішні функції: можна реалізувати за допомогою будь-якої мови програмування та у вигляді бібліотек користувацьких функцій (англ. UDF – User Defined Function) під'єднаних до сервера.

Набори символів: Firebird підтримує багато міжнародних наборів символів з багатьма варіантами сортування, зокрема типовий для українських користувачів Windows набір символів Win1251 підтримує три варіанти сортування, в тому числі win1251_ua, що дозволяє коректно сортувати отримані з бази дані з українськими символами на боці сервера. Окрім того, Firebird повністю підтримує Unicode, що дає можливість працювати з будь-якими наборами символів.

Firebird повністю підтримує стандарт SQL-92 та реалізує більшу частину стандартів SQL-99 і SQL-2003. Сюди входять вирази DML/DDL, об'єднання запитів, вирази UNION, DISTINCT, підзапити (IN, EXISTS), агрегатні функції (AVG, SUM, MIN, MAX, LIST (починаючи з версії 2.1)), вбудовані функції (ABS, CEIL, REPLACE, GEN_UUID тощо), обмеження цілісності (PRIMARY KEY, UNIQUE, FOREIGN KEY), та всі загальні типи даних SQL.

Особливості доступу та оброблення даних в Firebird

PSQL (Procedural SQL) – підмножина SQL в Firebird, за допомогою якої пишуться збережені процедури та тригери. Надає можливість програмісту

обробки даних у процедурному стилі – наприклад, за допомогою циклів.

```
CREATE OR ALTER PROCEDURE SOME_PROC(
    IN_ID INTEGER)
RETURNS (
    OUT_ID INTEGER)
AS
begin
   while (IN_ID < 10) do
    begin
     OUT_ID = IN_ID;
     IN_ID = IN_ID + 1;
     suspend;
    end
end
```

DSQL (Dynamic SQL) – підмножина SQL, за допомогою якої здійснюються запити до даних. Підтримуються неіменовані параметри.

```
select some_field1 from some_table where some_field2=? and some_field3
```

Доступ до баз даних Firebird може здійснюватися через розподілену бібліотеку доступу (dll або so, залежно від платформи) – такий спосіб є стандартним; сама бібліотека є в комплекті дистрибутиву. Також є можливість доступу через java і .net провайдери.

Бібліотека доступу реалізує набір функцій для маніпуляції з даними, які можуть бути експортовані і використані в будь-якій мові програмування.

Окрім того, є достатньо багато обгорток під різні мови програмування для цієї бібліотеки, що звільняють програміста від рутинної низькорівневої роботи. Так, для Delphi популярними є бібліотеки IBX, FibPlus, UIB. Для C++ – IBPP. Підтримка вбудована і в популярні скриптові мови, такі як PHP і Python.

Обмеження Firebird

Розмір бази даних – необмежено (залежить від можливостей файлової системи). Розмір таблиці бази даних – 32 ТБ. Максимальна ширина вибірки (сумарно всі поля; не враховуючи блоби; враховується фактичний розмір рядкових даних) – 64 Кб. Індексів на таблицю – 850. Довжина об'єкта метаданих (назва таблиці, процедури тощо) – 27 символів.

Апаратно-програмні вимоги та обмеження

Firebird існує у версіях для Unix (Linux, FreeBSD, Solaris, MacOS, HP-UX) та Windows і вимоги до апаратного забезпечення залежатимуть також від типу ОС, котра обслуговує сервер. Окрім того вимоги знаходяться в прямій залежності від очікуваного завантаження сервера баз даних, обсягу оброблюваних даних та кількості одночасно працюючих користувачів і говорити про конкретні цифри непросто. Проте загалом ці вимоги доволі низькі: при незначних навантаженнях та обсягах баз даних можна очікувати пристойної роботи на сервері з центральним процесором частотою 100–200 МГц та обсягом оперативної пам'яті 96-128 МБ.

4.2 Захист розробленого програмного забезпечення

Для організації надійної системи резервного копіювання потрібно скласти перелік інформаційних ресурсів, що підлягають захисту, пріоритезувати резервне копіювання критичної інформації, задати час зберігання і частоту створення копій, визначити заходи безпеки щодо припинення неавторизованого доступу, розрахувати необхідний розмір сховища для довгострокового і оперативного зберігання інформації.

Для забезпечення безпеки даних при знятті резервних копій практикується їх шифрування [16]. Зашифровані дані, як правило, займають більший обсяг, погано піддаються стисненню, а їх відновлення може тривати довше і вимагає більших обчислювальних ресурсів, що не зашифровані, проте воно дозволяє уникнути ризиків крадіжки або підміни інформації в репозиторії. Найчастіше шифрувати всі дані не потрібно, достатньо забезпечити захист найбільш критичної інформації.

При проектуванні систем резервного копіювання обов'язковим кроком є складання розкладу ротації резервних копій, що дозволяє гнучко розподілити ресурси системи резервного копіювання. Так, оперативна копія може зберігатися

на серверах з високою доступністю, архівні копії – на сторонніх носіях: оптичних дисках або на допоміжних серверах. Кількість оперативних і архівних копій слід розрахувати з урахуванням вимог до часу відновлення і тривалості зберігання інформації. Крім того, слід враховувати доступний обсяг системи зберігання даних, визначеної для архіву й сховища.

Для визначення вимог до технічної бази системи резервного копіювання може застосовуватися підхід, який лежить в основі систем управління життєвим циклом інформації (Information Lifecycle Management, ILM), який постулює динамічність цінності інформації на основі її актуальності в бізнес-процесах. З його допомогою можна відокремити критичні дані, що вимагають високої швидкості доступу, частого резервування і швидкого відновлення, від простіших і менш критичних даних, розташувавши їх на різних технічних майданчиках.

Параметри, що визначають надійність[17]:

1. Безвідмовність – властивість об'єкта безупинно зберігати працездатний стан протягом деякого часу або напрацювання.

2. Ремонтопридатність – властивість об'єкта пристосовуватись до підтримання та відновлення працездатного стану.

3. Довговічність – властивість об'єкта безупинно зберігати працездатність від початку експлуатації до настання граничного стану.

4. Збереженість – властивість об'єкта зберігати працездатність протягом усього періоду зберігання та транспортування.

5. Живучість – властивість об'єкта зберігати працездатність при відмові окремих функціональних вузлів.

Захист розробленого програмного забезпечення відбувається за допомогою алгоритму Lucifer.

Алгоритм Lucifer являє собою мережу перестановок і підстановок, його основні блоки нагадують блоки алгоритму DES. В DES результат функції f складається операцією XOR із входом попереднього раунду, утворюючи вхід наступного раунду. В S-блоках алгоритму Lucifer 4-бітові входи й виходи, вхід S-

блоків являє собою перетасований вихід S-блоків попереднього раунду, входом S-блоків першого раунду служить відкритий текст. Для вибору використовуваного S-блоку із двох можливих використовується біт ключа. (Lucifer реалізує все це в єдиному T-блоці з 9 бітами на вході й 8 бітами на виході). На відміну від алгоритму DES, половини блоку між раундами не переставляються, та й саме поняття половини блоку в алгоритмі Lucifer не використовується. У цього алгоритму 16 раундів, 128-бітові блоки й більше проста, чим в DES, схема розгорнення ключа.

Блок тексту розглядається як ненегативне ціле число, або як кілька незалежних ненегативних цілих чисел. Довжина блоку завжди вибирається рівною ступеню двійки. У алгоритмі Lucifer використовуються наступні типи операцій:

– Таблична підстановка, при якій група біт відображається в іншу групу біт. Це так звані S-box.

– Переміщення, за допомогою якого біти повідомлення переупорядковуються.

– Операція додавання по модулю 2, позначувана XOR або $\oplus$.

– Операція додавання по модулю 2^{32} або по модулю 2^{16}.

– Циклічне зрушення на деяке число біт.

Ці операції циклічно повторюються в алгоритмі, створюючи так звані раунди. Входом кожного раунду є вихід попереднього раунду й ключ, що отриманий по певному алгоритму із ключа шифрування K. Ключ раунду називається підключем. Алгоритм шифрування може бути представлений у такий спосіб, як наведено на рисунку 4.3. Вихідне незашифроване повідомлення надходить до Раунд 1, після цього отримуємо результат шифрування першого раунда і т.д. до Раунд N. Після чого отримуємо зашифрований текст.

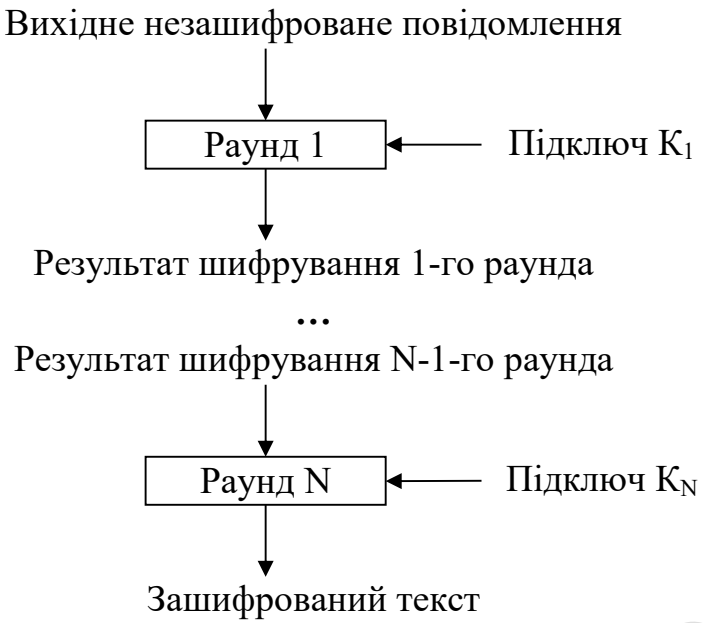

Рисунок 4.3 – Структура алгоритму Lucifer

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На сучасному рівні розвитку бізнесу інформація набуває величезну цінність. У корпоративному сегменті доступність та збереженість даних визначає безперервність протікання бізнес-процесів та ефективна взаємодія з цільовою аудиторією. Впровадження систем резервного копіювання даних забезпечує можливість оперативного відновлення інформації в разі: фізичної відмови пристроїв зберігання або серверів; втрати обладнання на час проведення перевірок державними або силовими структурами; відмова в роботі операційної системи або програмного забезпечення, що несуть за собою порушення цілісності даних; зловмисного або випадкового видалення корпоративної інформації в результаті відсутності політики зберігання даних та розмежування прав доступу; адресної хакерської атаки та дій шкідливих програм.

На рисунку 5.1 зображено головне вікно розробленої у випускній кваліфікаційній роботі за другим магістерським рівнем системи резервного копіювання даних хмарного, віртуального й фізичного середовища. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

- Функціональних кнопок ПЗ.
- Навігаційного меню, яке викликається натисканням правої клавіші маніпулятора миші.
- Розділу обрання групи.
- Розділу виведення результату роботи системи.

Розроблена програма має простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей опанує й цю програму.

Система не вимагає яких-небудь спеціальних знань і розрахована як на

системних адміністраторів, так і на простих користувачів, що мають мінімальний досвід володіння ПК.

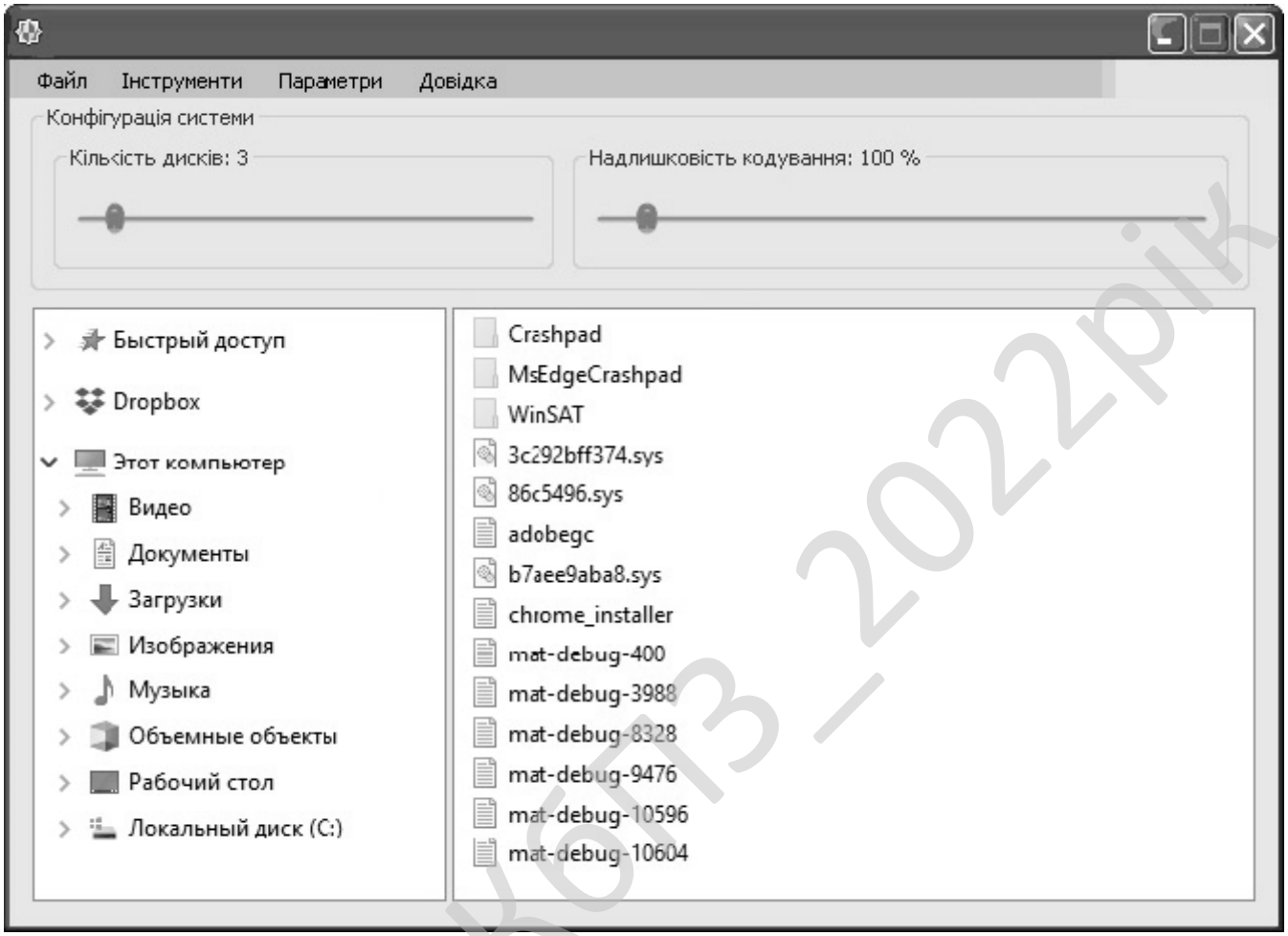

Рисунок 5.1 – Головне вікно ПЗ

Досить встановити програму і створити в ній хоча б одне завдання, в якому налаштувати розклад для автоматичного бекапа.

Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше необхідно слідувати інструкціям, які пропонує програма. Передбачено обробку помилок при виняткових ситуаціях, відмовостійкість при розривах зв'язку з віддаленим ПК.

Для компаній різного розміру буде оптимальним обчислення періодичності створення систем резервного копіювання виходячи з вартості бізнес-часу і комерційних даних за один день. В середньому досить створювати:

раз на місяць – повну резервну копію; раз в тиждень – диференційну копію; раз в день – інкрементальну копію.

Програмний продукт дозволяє не тільки створювати резервні копії, але і виконувати їх обслуговування. Мінімізувати кількість операцій, які необхідно виконувати вручну, допомагають виконати інтерактивні модулі – майстер налаштування. Вони дозволяють робити дубльовані копії (наприклад, в хмару), дозволяють автоматизувати всі можливі кроки. У них вже записані шаблони завдань і типові конфігурації. Їх залишається тільки вибрати і зробити їх налаштування з урахуванням потреб, ресурсів і специфіки роботи компанії.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

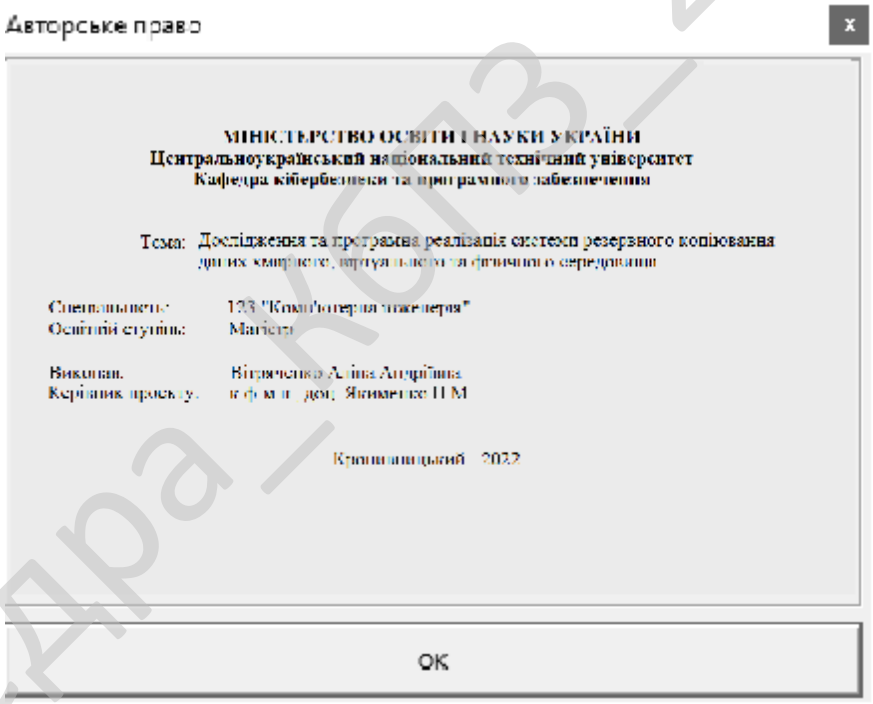

Рисунок 5.2 – Вікно програми з авторськими даними

Ключовим моментом розвитку сучасних засобів резервного копіювання, стало впровадження хмарних технологій. Використання цих технологій робить більш простим управління бекапів і забезпечує доступ до них у будь-який час.

За останні пару років хмарні технології отримали колосальний розвиток, і набрали величезну популярність, внаслідок появи безлімітного інтернету і розвитку мобільних мереж.

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

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

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

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

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

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

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

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

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

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

– У програмі можуть бути пропущені деякі маршрути.

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

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

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

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

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

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

Обрано умови розповсюдження – Freeware. Це власницьке програмне забезпечення, котре можна безоплатно використовувати протягом необмеженого терміну без обмежень у функціональності, і поширюване без сирцевих кодів. Автори такого програмного забезпечення, як правило, хочуть «дати щось спільноті», але хочуть також контролювати його подальшу розробку. Іноді, коли програмісти вирішують припинити розробку, вони передають сирцевий код іншим програмістам, або ж спільності як вільне програмне забезпечення. Дуже часто плутають поняття «безкоштовне програмне забезпечення» та «вільне програмне забезпечення», хоча вони суттєво відрізняються.

6 НАУКОВА НОВИЗНА

Під час виконання дослідження та реалізації системи резервного копіювання даних хмарного, віртуального і фізичного середовища, були розглянуті аналогічні існуючі системи, архітектури та програмні рішення. Більшість з яких має такий недолік, як платна підписка, тобто постійне користування вимагатиме затрат, і завчасного продовження цієї підписки. Аналіз однотипних безкоштовних застосунків показав, що велика кількість систем складні для розуміння простому користувачу. Таким чином, розробка власного додатку буде забезпечувати максимальну простоту користування, як для системних адміністраторів так і звичайних користувачів, що мають мінімальний досвід роботи з ПК.

Основним завданням дослідження є запобігання втрати даних завдяки проведенню резервного копіювання. Захист даних в такий спосіб є простим та надійним, що буде зручно у використанні.

Було досліджено методи і моделі для резервного копіювання та після аварійного відновлення серверних даних. Дослідження стану проблеми показало, що стрімкий зріст обсягів сховищ даних пов’язаний із потребами бізнесу. Водночас ІТ-компанії визнають незадовільним стан інфраструктури для здійснення резервних копій важливої інформації. Моніторинг стану проблеми та наявних методів і моделей резервного копіювання та після аварійного відновлення підтвердив необхідність розробки більш простих додатків. У роботі запропонований додаток, який вирішить усі перелічені проблеми та буде корисним при резервному копіюванні інформації.

Застосування хмарних технологій робить більш простим управління бекапів і забезпечує доступ до них у будь-який час на відміну від існуючих аналогів.

7 ЕКОНОМІЧНА ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми дипломного проекту

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В випускній кваліфікаційній роботі було проведене дослідження та виконана програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища.

Розроблене програмне забезпечення має достатню надійність і задовольняє всім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позна-чення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	280
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	1
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	2
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	2
17. Складність кінцевого програмного продукту (1-6)	–	2
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	2
20. Вимоги до швидкодії ПП (1-6)	–	2
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	2
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	2
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	28000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \text{ Size}^B, \quad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \quad (7.2)$$

де: Wi – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \quad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 6{,}78 \cdot (0{,}88 \cdot 0{,}93 \cdot 0{,}88 \cdot 0{,}91 \cdot 0{,}95 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 1{,}22 \cdot 1{,}16 \cdot 1{,}1 \cdot 1{,}1 \cdot$$
$$\cdot 1{,}12 \cdot 1{,}1 \cdot 1 \cdot 1{,}1 \cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \quad (7.4)$$

де: C – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 3{,}23 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 52 = 106 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до

таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	106	Ф 7.1-7.4
Впровадження	13	Д13
Всього	147	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{147 \cdot 1}{60 - 5} = 2,7 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо

до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системни блок ПК	90	7	630	10,5
Монітор	60	7	420	7
Клавіатура	30	7	210	3,5
Маніпулятор «мишка»	30	7	210	3,5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор–маршрутизатор	30	1	30	0,5
Кабельні господарства ЛВС на 1 м. п.	2,5	100	250	4,17
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	34,83

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків не повинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_{ч} \cdot n_{тіс}}{1{,}2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{35 \cdot 3}{1{,}2} = 87{,}5 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$$Ч_{ел} = 87{,}5/(60 \cdot 8) = 0{,}2 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2012 R2, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Розміщення графіки і контенту на Інтернет сторінках	0,5	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний

розклад.

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	0,25	13672	10254
Продакт-менеджер	0,25	10000	7500
Інженер-програміст	2,7	12500	101250
Інженер-електронник	0,2	10000	6000
Інженер-системотехнік	0,25	10000	7500
Адміністратор мережі	0,5	10000	15000
Системний програміст	0,25	10000	7500
Дизайнер WEB	0,25	10000	7500
Інженер-верстальник	0,25	10000	7500
Бухгалтер-економіст	0,5	10000	15000
Всього за період розробки	$R_{сп}$ = 5,4	-	$Ф_{роб}$ = 185004

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рq}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{185004}{5,4 \cdot 60} = 571 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного

квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць;

$S_у$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались за прайсом фірми Компбест за 06.11.22 – джерело https://compbest.com.ua/.

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		10947
Системний блок		7347
Процесор	Intel Core i5-4590 (4 ядра по 3.3-3.7GHz), 6 MB cache	1750
Системна плата	MSI H81M-P33 6 x USB 2.0, 2 x USB 3.0, VGA, DVI, 2 x PS/2, LAN (RJ-45), 5 x Audio, FireWire	1200
Відеокарта	nVidia GeForce GTX 660, 2 GB GDDR5, 192-bit	750
Жорсткий диск	240 GB SSD	1200
Оперативна пам’ять	Samsung DDR3-1333 8Gb PC3-10600R ECC Registered (M393B1K70CH0-CH9Q5)	900
DVD-привод	DVDRW Pioneer DVR-TD10RS SATA Slim Black Bulk (DVR-TD10RS)	416
Корпус	ATX Middle Tower FOXCONN Pro, 3GTLA-489, PSU 350W(FSP Brand: ATX-350PNR, 12cm), black, (front bezel – black+light silver; body material – 0.6mm), 80mm fan (rear), 2xUSB2.0/AUDIO/MIC, Air Duct, Tool-less chassis design,Thermally Advantaged Chassis	911

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Кулер	–	–
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black	220
Інше	Клавіатура, мишка	Подарунок
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1, 170/160, D-SUB, Wide)	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	15	10947	16420,5	180625,5
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	-	-	-	0
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	199177

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400
Група 4			
3. Обчислювальна техніка	199177	-	-
Всього по групі	199177	50	99588,5
Група 5, 6			
4. Вимірювальні пристрої	5190	25	1297,5
5. Транспортні засоби	0	20	0,0
6. Господарський інвентар	28000	25	7000
Всього по групі	33190	-	8297,5
7. Нематеріальні активи	120000	10	12000
Разом	$К_р$ = 1760367		$А_р$ = 190286

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot T_{пз}}{N_e}, \quad (7.11)$$

де: N_e – кількість екземплярів програм, шт.

$$З_о = 571 \cdot 147/280 = 300 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 300 \cdot 10 \cdot 0{,}01 = 30 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: $Н_с$ – відрахування на соціальні потреби, %.

$$С_{оц} = 0{,}01 \cdot 22(300+30) = 73 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Н_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 300 \cdot 15 \cdot 0{,}01 = 45 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картріджей, тонеру, грн.;

$N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 0,5 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=210 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 210 \cdot 0,5 = 105 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 100):

$$З_{М2} = \sum Ц_д \,, \tag{7.17}$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 21,5 грн./шт., DVD-R box – 33,6 грн./шт.

$$З_{М2} = 100 \cdot 33,6 = 3360 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \tag{7.18}$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574+558+570 = 1702 \text{ грн.}$$

$$З_М = (105 + 3360 + 1702)/280 = 18 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0,01, \tag{7.19}$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 300 \cdot 15 \cdot 0,01 = 45 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 280 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \tag{7.20}$$

де: $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 190286 \cdot 3/(280 \cdot 12) = 170 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$C_п = З_о + З_д + C_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$C_п = 300+30+73+45+18+45+170 = 681 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1. Основна зарплата виконавців	$З_о$	300
2. Додаткова зарплата виконавців	$З_д$	30
3. Відрахування на соціальні потреби	$C_{оц}$	73
4. Загальногосподарські витрати	$Г_{осп}$	45
5. Витрати на матеріали	$З_М$	18
6. Освоєння нових операційних систем, мов програмування	$О_п$	45
7. Амортизація основних фондів	$А_м$	170
8. Повна собівартість програмного забезпечення	$C_п$	681
9. Плановий прибуток	$П_р$	341
10.Ціна підприємства $Ц_п = C_п + П_р$	$Ц_п$	1022
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	204,4
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	1226,4

Визначимо плановий прибуток за рівнем рентабельності (P_n) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 50%.

$$П_р = 0{,}01 \cdot P_n \cdot C_n, \quad (7.22)$$

де: P_n – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 50 \cdot 681 = 341 \text{ грн.}$$

7.6 Визначення об'єму капітальних вкладень у споживача програмної продукції

Об'єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.9.

Таблиця 7.10 – Розрахунок об'єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	1226
Всього капітальних витрат	–	1226

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Витрати на профілактичні роботи:

$$З_р = T_р \cdot З_г \cdot (1 + 0{,}01 \cdot H_q) \cdot (1 + 0{,}01 \cdot H_c), \quad (7.23)$$

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на обслуговування системи	$З_р$	15190	6500
2. Витрати на електроенергію	$З_{ел}$	1488	1030
3. Витрати на амортизацію	$З_{ам}$	0	307
Всього витрат за рік	$І$	16678	7837

де: $Т_р$ – кількість годин обслуговування кожного комп'ютера за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, $грн/год$.

Після купівлі нового програмного забезпечення витрати на обслуговування системи зменшились з 15190 грн до 6500 грн на рік.

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	1226	–	306,5
Всього відрахувань	-	–	1226	–	306,5

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

$$З_{ел\ баз} = 0{,}545 \cdot 1300 \cdot 2{,}1 = 1487{,}85 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}545 \cdot 900 \cdot 2{,}1 = 1030{,}05 \text{ грн.}$$

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - С_п) \cdot N_е - \sum_{i=1}^{m} E_{р_m} \cdot К_{р_m}, \quad (7.25)$$

де: $К_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (1022 - 681) \cdot 280 - (0{,}05 \cdot 1408000 + 0{,}5 \cdot 199177 +$$
$$+0{,}25 \cdot 33190 + 0{,}1 \cdot 28000) \cdot 3/12 = 50208 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$Т_в = \frac{К_р}{(Ц_п - С_п) \cdot N_е}, \quad (7.26)$$

де: $К_р$ – балансова вартість основних фондів розробника.

$$Т_в = \frac{1760367}{(1022 - 681) \cdot 280 \cdot 12 / 3} = 4{,}5 \text{ роки}$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (І_б - І_н) - E_н (К_н - К_б), \quad (7.27)$$

де: $І_б$, $І_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$К_б$, $К_н$ – об’єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (16678\text{-}7837)\text{-}0{,}25 \cdot 1226 = 8535 \text{ грн.}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	280
2. Повна собівартість розробленої програми	Грн.	681
3. Ціна розробленої програми	Грн.	1022
4. Плановий прибуток від реалізації розробленої програми	Грн.	341
5. Рентабельність програмної продукції	%	50
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1760367
7. Загальний прибуток від реалізації програмної продукції	Грн.	95480
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	50208
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Рік	4,5
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	1226
11.Величина економічного ефекту у користувача програмної продукції	Грн.	8535
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Рік	0,14

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \qquad (7.28)$$

$$T_{сп} = \frac{1226}{16678 - 7837} = 0{,}14 \ року .$$

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, а також підвищується якість приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Вступ

Законом України “Про охорону праці” [51] 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» [52], 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 отримують негативний вплив на органи зору, а також мають значну розумову напругу і нервово-емоційне навантаження. Руки (суглоби пальців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаження. 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 [52], 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:

- 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.

Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальна машина (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Тому що робота програміста характеризується істотним зоровим навантаженням, це вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно створити належний мікроклімат, праметри якого регламентуються Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

- ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.

- ризик виникнення пожежі;

- негативний вплив на органи зору людини;

- ризики ураження електричним струмом;

- недостатня, або надмірна освітленість робочого місця;

- електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання (коливання);

- несприятливі мікрокліматичні умови;

- нервово-емоційна напруженість праці;

- інтелектуальні навантаження;

- монотонність праці;

- невідповідність ергономічних показників робочого місця діючим вимогам;

- шум;

- статичні навантаження на кістково-м’язовий апарат.

Дослідження та 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:

Таблиця 8.1 - Розміри приміщення

Найменування	Значення, м
Ширина	10,23
Довжина	9,2
Висота	3,55

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	47
Об'єм, V	$м^3$	не менше 20.0	167

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

У зазначеному приміщенні працюють двоє людей. 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 відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [52], 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» [52] 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.

Таблиця 8.3 - Оптимальні й фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Темпе-ратура, °С	Вологість, %	Швид-кість повітря, м/с	Темпе-ратура, °С	Воло-гість, %	Швид-кість повітря, м/с
Холодна	22-24	40-60	0,1	21-22	45-50	0,1
Тепла	23-25	50-70	0,1	22-23	55-60	0,11

У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

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 приміщення.

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 [53], 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 [53], 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. [53].

Крім того, все поле зору має бути освітленим достатньо рівномірно - це основна гігієнічна вимога. Тому що 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.

Розробка заходів з умов поліпшення охорони праці

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

- розміри приміщення, у розрахунку на одного 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).

Регулярне 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.

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

Розрахункова частина

Початкові дані для розрахунку захисного штучного заземлення: опір заземлювача, який нормується: $R_{ЗН} = 4$ *Ом*.

Для захисного штучного заземлення застосовуються вертикальні електроди з металевого прутка діаметром 30 мм (*D=30 мм=0,03 м)* довжиною *L*=3 *м* та горизонтальний електрод — металева смуга з перетином 40*4 мм, тип ґрунту — глина (питомий опір 40 Ом*м). Відстань між вертикальними заземлювачами (електродами) *A*=3 *м.*

Глибина закладення горизонтального контура заземлення *t= 0,8 м*.

Умовна товщина верхнього шару ґрунту: *H=0,4 м*. Напруга — 220/380 В. Розрахункова схема розташування заземлюючих електродів — у ряд.

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача.

Потрібно визначити необхідну кількість вертикальних заземлювачів та довжину смуги (горизонтального заземлювача).

Розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,8+3/2=2,3 \text{ м.}$$

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

$$\rho = \psi \rho_2 = 1,36*40=54,5 \text{ Ом*м,}$$

де $\psi = 1,36$ - табличне значення коефіцієнта сезонності для відповідної кліматичної зони у багатошаровому грунті [54];

$\rho_1 = 50$ *Ом*м* - табличне значення питомого опору верхнього шару грунту [55];

$\rho_2 = 40$ *Ом*м* - табличне значення питомого опору нижнього шару грунту [54].

Опір розтіканню електричного струму одного електрода вертикального заземлювача [8]:

$$R_0 = 0{,}366\frac{\rho}{L}\left(\lg\frac{2L}{D} + \frac{1}{2}\lg\frac{4T+L}{4t-L}\right) = 0{,}366\frac{54{,}5}{3}\left(\lg\frac{4\cdot 2{,}3+3}{4\cdot 2{,}3-3}\right) = 16{,}27\ \text{Ом}$$

Відношення *A/L=3/3=1.*

Визначаємо коефіцієнт екранування вертикальних електродів *Кев=0,8* при попередній (орієнтовній) кількості вертикальних електродів, яке дорівнює 4 [54].

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

$$N=R_O /(Кев\ R_{ЗН})= 20{,}1 / (0{,}8*4)=5{,}87 \approx 6\ шт.$$

Визначаємо довжину з'єднуючої смуги [8]:

$$L_П =1{,}05*A*N= 1{,}05*3*6=18{,}8\approx 19\ м$$

Опір розтіканню електричного струму з'єднуючої смуги [54]:

$$R_П =0{,}366(\rho_2*K_П/L_П)lg(2(L_П *L_П)/(К*t))=$$

$$=0{,}366(40*5/16)*[lg(2*16*16)/(0{,}04*0{,}8)]=16{,}5\ Ом$$

де $K_П=5$ - табличне значення коефіцієнта сезонності для відповідної кліматичної зони з'єднуючої смуги: [54].

Загальний опір розтіканню електричного струму заземлювача [54]:

$$R =(R_O*R_П)/(R_O*\eta_П+ N*R_П*Кев)=$$

$$=(16{,}27* 16{,}5)/(16{,}27* 0{,}75+ 6* 16{,}5* 0{,}8)=2{,}93\ Ом.$$

де $\eta_П = 0{,}75$ - табличне значення коефіцієнта екранування з'єднуючої смуги [54].

Умова $R \le R_{ЗН}$ виконується ($2{,}93 \le 4$).

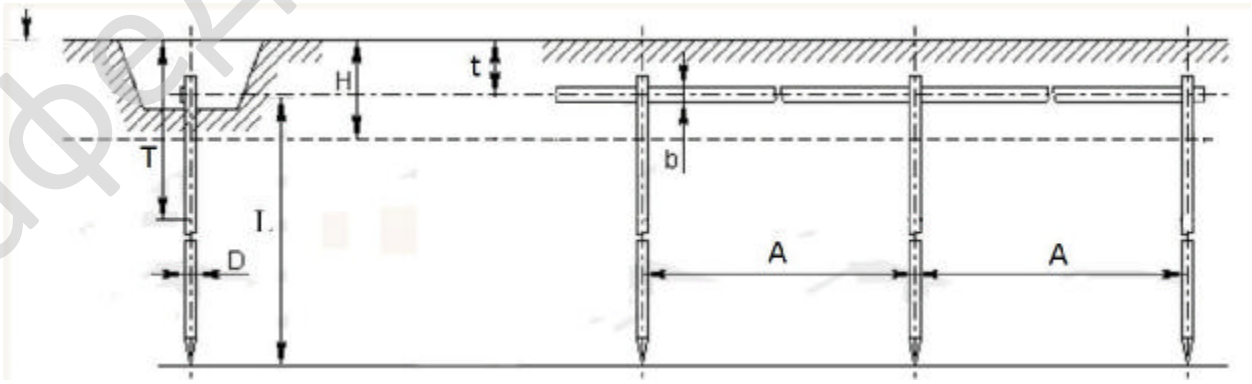

Рисунок 8.1 - Схема штучного заземлення у приміщенні

Висновки до розділу

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

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

9 ОСНОВНІ ВИСНОВКИ

Дослідження та програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим магістерським рівнем, призначено для системи резервного копіювання даних хмарного, віртуального й фізичного середовища.

Тема резервного копіювання настільки ж важлива, наскільки й обширна. Вона охоплює багато аспектів взаємодії людини й обчислювальної техніки, забезпечуючи захист від псування або втрати інтелектуальної праці або інших даних, що зберігаються в цифровому вигляді.

Вітчизняні розробки в цій області представлені в недостатній мірі.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем резервного копіювання даних.

– Досліджена система резервного копіювання даних хмарного, віртуального й фізичного середовища.

– На основі отриманих результатів досліджень створена програмна реалізація системи резервного копіювання даних.

Розроблені під час виконання кваліфікаційної бакалаврської роботи алгоритми дозволяють успішно вирішувати завдання резервного копіювання даних хмарного, віртуального й фізичного середовища.

Розроблене програмне забезпечення має простий, дружній інтерфейс для користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, й 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#. Visual C# – жорстко типізована об’єктно-орієнтована мова, призначена для розробки

різноманітних безпечних і потужних додатків, що виконуються у середовищі .NET Framework. Ця мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи резервного копіювання даних. Це дозволило мінімізувати термін розробки програмного забезпечення, і, як наслідок, 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.

Даються необхідні рекомендації з установки та використання розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Lucifer.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Програми резервного копіювання даних. 02.2021. [Електронний ресурс] – Режим доступу: https://ukr.kagutech.com/4114827-backup-software

2. Acronis True Image. 2020. [Електронний ресурс] – Режим доступу:https://www.acronis.com/ru-ru/support/documentation/ATI2020/index.html#16515.html

3. Norton Ghost 15.0 – Руководство пользователя. [Електронний ресурс] – Режим доступу: https://docplayer.ru/50039452-Norton-ghost-rukovodstvo-polzovatelya.html

4. Back2zip 125 бесплатная программа для резервного копирования данных. [Електронний ресурс] – Режим доступу: http://www.softfly.ru/sistema/rezervnoe-kopirovanie/39-back2zip-125

5. Comodo BackUp Free 5GB. [Електронний ресурс] – Режим доступу: https://www.comodorus.ru/free_versions/detal/comodo_free/12

6. Обзор систем резервного копирования и восстановления данных 22.02.2019. [Електронний ресурс] – Режим доступу: https://www.computerra.ru/235234/obzor-sistem-rezervnogo-kopirovaniya-i-vosstanovleniya-dannyh

7. Система резервного копирования. 2010/04/30. [Електронний ресурс] – Режим доступу: https://www.tadviser.ru/index.php/Статья:Система_резервного_копирования

8. Альбекова З.М., Ахвердов А.А., Кирпиченко В.А. Резервное копирование данных с помощью облачных сервисов при управлении данными в сетях. 2018. [Електронний ресурс] – Режим доступу: https://cyberleninka.ru/article/n/rezervnoe-kopirovanie-dannyh-s-pomoschyu-oblachnyh-servisov-pri-upravlenii-dannymi-v-setyah

9. Резервное копирование для малого бизнеса — это просто! 30.11.2020. [Електронний ресурс] – Режим доступу: https://ko.com.ua/rezervnoe_kopirovanie_dlya_malogo_biznesa_jeto_prosto_135495

10. Chernyakov A.V. DATA BACKUP SYSTEM PLANNING. 2017. [Електронний ресурс] – Режим доступу: https://cyberleninka.ru/article/n/proektirovanie-sistemy-rezervnogo-kopirovaniya-dannyh/viewer

11. Джоэл Берман. Backup для чайников, 2014. [Електронний ресурс] – Режим доступу: http://2015.russianinternetforum.ru/upload/acronis-book.pdf

12. Система резервного копирования. 2019. [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/421251/

13. Pete McLean "An Introduction to RAID Redundant Arrays of Inexpensive Disks. Digital Equipment Corporation, April 1991.

14. Azer Bestaves "IDA-based Redandant Arrays of Inexpensive Disks." Proceed. of I-st International Conference on Parallel and Distributed Information Systems. Dec 4-6, 1991. IEEE Comp. Society Pkss.

15. Lee, Edward Kiehyen. "Performance Modeling and Analisis of Disk Arrays" University of Califonia, Berkely, Report No. UCB/CSD-93-770, September 1993.

16. Традиционные методы и средства защиты данных, реализованные в базе данных с универсальной моделью данных [Текст] / Л. С. Сорока, В. И. Есин, М. В. Есина // Академія митної служби України. Вісник Академії митної служби України. Серія "Технічні науки". 2010 р. № 2 (44) / Академія митної служби України. – Д., 2010. – С. 7

17. Кашев Д.Е., Чугунов М.М. Повышение надежности ИТ-инфраструктуры при использовании виртуализации, 2013. [Електронний ресурс] – Режим доступу: https://cyberleninka.ru/article/n/povyshenie-nadezhnosti-it-infrastruktury-pri-ispolzovanii-virtualizatsii

18. Есин В. И. Безопасность информационных систем и технологий / Есин В. И., Кузнецов А. А., Сорока Л. С. – Х. : ЭДЭНА, 2010. – 656 с.

19. Полтавцева М. А., Хабаров А. Р. Безопасность баз данных: проблемы и перспективы Международный журнал Программные продукты и системы №3 2016. [Електронний ресурс] – Режим доступу: http://www.swsys.ru/index.php?page=article&id=4175

20. Терентьев А.М. Информационная безопасность в крупных локальных сетях. – «Концепции», N1(9), 2002, с. 25-30.

21. Смирнов А.А. Разработка методики оценки среднего времени обслуживания информационных пакетов в телекоммуникационной сети / А.А.Смирнов, В.В.Босько, Е.В.Мелешко // Системи управління, навігації та зв'язку. – Київ: ДП «Центральний науково-дослідний інститут навігації і управління», 2009. – Вип. 2(10). – С.162-165.

22. Смирнов А.А. Усовершенствование метода управления очередями в многопротокольных узлах телекоммуникационной сети / А.А.Смирнов, Е.В.Мелешко // Збірник тез та доповідей другої всеукраїнської науково-практичної конференції «Системний аналіз. Інформатика. Управління». Запоріжжя. Тези доповідей. Запоріжжя: КПУ, 2011. – 193-195 с.

23. Elwalid, D. Mitra, I. Saniee, and I. Widjaja. Routing and Protection in GMPLS Networks: From Shortest Paths to Optimized Designs // Journal of lightwave technology. – 2003. – №21(11), P. 2828-28-38.

24. A.B. Bagula, M. Botha, and A.E Krzesinski. Online Traffic Engineering: The Least Interference Optimization Algorithm // IEEE Communications Society – 2004, P. 1232-1236.

25. Basabi Chakraborty. Simultaneous Search for Multiple Routes using Genetic Algorithm / IEEE lntemational Conference on Computational oltelligencs for Measurement System and Applications Boston. MA, USA, 14-16, July 2004, P. 77-80.

26. Chiara Francalanci and Paolo Giacomazzi. High-Performance Self-Routing Algorithm for Multiprocessor Systems with Shuffle Interconnections // IEEE Transactions on parallel and distributed systems. – 2006. – №1, P. 38-50.

27. Chris Loeser, Andre Brinkmann, Ulrich Ruckert. Distributed Path Selection (DPS) a Traffic Engineering Protocol for IP-Networks / Proceedings of the 37th Hawaii International Conference on System Sciences – 2004, P. 1-8.

28. Gaurav Barot, Chintan Mehta et al. Hadoop Backup and Recovery Solutions Paperback – July 1, 2015, 206c.

29. Сидоренко В.В., Константинова Л.В., Смірнов С.А. Організація баз даних Навчальний посібник. – Кропивницький: ЦНТУ, 2018. – 274 с.

[Електронний ресурс] – Режим доступу: http://dspace.kntu.kr.ua/jspui/bitstream/123456789/10527/1/NavPosOBD.pdf

30. Desrochers, S. Mohseni On determining the structure of non-linear systems // Int. J. Control. 1984. V. 40. N 5. — P. 923-938.

31. Elwalid, D. Mitra, I. Saniee, and I. Widjaja. Routing and Protection in GMPLS Networks: From Shortest Paths to Optimized Designs // Journal of lightwave technology. – 2003. – №21(11), P. 2828-28-38.

32. A.B. Bagula, M. Botha, and A.E Krzesinski. Online Traffic Engineering: The Least Interference Optimization Algorithm // IEEE Communications Society – 2004, P. 1232-1236.

33. An Chen, Albert Kai-Sun Wong, and Chin-Tau Lea, Senior Member. Routing and Time-Slot Assignment in Optical TDM Networks // IEEE Journal on selected areas in communications. – 2004. – №22(9), P. 1648-1657.

34 Andrew B. Kahng, Stefanus Mantik, and Dirk Stroobandt. Toward Accurate Models of Achievable Routing // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2001. – №20(5), P. 648-659.

35. Anees Shaikh, Jennifer Rexford, and Kang G. Shin. Evaluating the Impact of Stale Link State on Quality-of-Service Routing // IEEE/ACM Transactions on Networking. – 2001. – №9(2), P. 162-176.

36. Basabi Chakraborty. Simultaneous Search for Multiple Routes using Genetic Algorithm / IEEE lntemational Conference on Computational oltelligencs for Measurement System and Applications Boston. MA, USA, 14-16, July 2004, P. 77-80/

37. LeBaron "A Fast Algorithm for the BDS Statistic", Studies in Nonlinear Dynamics and Econometrics. 1997. Vol. 2. No. 2. P. 53-59.

38 Barakat, E. Altman, and W. Dabbous. On TCP performance in a heterogeneous network: a survey // IEEE Communications Magazine. – 2000. – №38(1). – P. 40 – 46.

39. Casetti, R. Lo Cigno, M. Mellia, M. Munafo. A New Class of QoS Routing Strategies Based on Network Graph Reduction // Proceedings of IEEE INFOCOM. – 2002, P.715-722.

40. Chiara Francalanci and Paolo Giacomazzi. High-Performance Self-Routing

Algorithm for Multiprocessor Systems with Shuffle Interconnections // IEEE Transactions on parallel and distributed systems. – 2006. – №1, P. 38-50.

41. Chris Loeser, Andre Brinkmann, Ulrich Ruckert. Distributed Path Selection (DPS) a Traffic Engineering Protocol for IP-Networks / Proceedings of the 37th Hawaii International Conference on System Sciences – 2004, P. 1-8.

42. Dai Boong Lee and Hwangjun Song. Dynamic Class Selecting Mechanism for Guaranteed Service with Minimum Cost over Relative Differentiated-Services Networks / IEEE International Conference on Multimedia and Expo (ICME) – 2004, P. 237-240.

43. Dan Pei, Dan Massey, Lixia Zhang. Detection of Invalid Routing Announcements in RIP Protocol // IEEE GLOBECOM – 2003, P. 1450-1455.

44. Donna Ghosh, Venkatesh Sarangan, and Raj Acharya. Quality-of-Service Routing in IP Networks // IEEE Transactions on Multimedia. – 2001. – 3(2), P. 200-208.

45. Gang Cheng and Nirwan Ansari. A New Heuristics For Finding The Delay Constrained Least Cost Path // IEEE GLOBECOM – 2003, P. 3711-3715.

46. Gang Cheng, Li Zhu, and Nirwan Ansari. A New Deterministic Traffic Model for Core-Stateless Scheduling // IEEE Transactions on communications. – 2006. – № 4, P. 704-713.

47. G.Kreisselmeier A robust indirect adaptive control approach // Int. J. Control. 1986. Vol. 43. № 1. — P. 161–175.

48. Модели услуг IBM Cloud: IaaS, PaaS и SaaS [Електронний ресурс] – Режим доступу: https://www.ibm.com/ru-ru/cloud/learn/iaas-paas-saas

49. D. Patterson G. Garth, R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)" University of Califomia, Berkely, Report No. UC8/CSD/87/391, December 1987.

50. Lee, Edward Kiehyen; Chen, Peter Ming-Chien, and others "RAID-II A Scalable Storage Architecture for High-Bandwith Network File Service" University of California, Berkely, Report No. UCB/CSD-92-672, October 1992.

51. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 16.06.2022).

52. Наказ Міністерства соціальної політики України 14.02.2018 № 207

«Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508 (дата звернення: 16.06.2022).

53. Державні будівельні норми України: ДБН В.2.5-28:2018. - Режим доступу до ресурсу: https://goo.su/9AkQ (дата звернення: 16.06.2022).

54. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : "Колос", 1973. – 238 с.

55. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. - Кіровоград: КІСМ, 1997. - 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358 (дата звернення: 16.06.2022).

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРМ-123.22.0005.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		ВітряченкоА.			Дослідження та програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища	Літ. Б	Аркуш 1 / Аркушів 6
Перевірив		Якименко Н.М.					
Н. Контр.		Гермак В.С.				ЦНТУ КІ-21М	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи резервного копіювання даних хмарного, віртуального й фізичного середовища.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 19-13 від 17.08.2022 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за (другим) магістерським рівнем освіти є дослідження та програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем освіти є стосовна до теми література й аналогічні системи, що існують.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частини системи резервного копіювання, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи резервного копіювання.

5.2 Показники призначення

Система повинна забезпечувати:

– резервне копіювання даних хмарного, віртуального й фізичного середовища;

– 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 за всіма правилами, 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. за Цельсієм;

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище .NET Framework. Мова 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 вигляді 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 розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2022 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 –103 аркуші.

10 Етапи розробки

10.1 Збір і обробка інформації за темою 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 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки й 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 10.12.2022 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем освіти на захист 23.12.2022 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

_____ Якименко Н.М.

Дослідження та програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 64

Літера: РП

Кропивницький – 2022 року

Файл FileAnalyzer.cs - контроль цілісності даних

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

namespace RecoveryDisk
{
    /// <summary>
    /// Клас контролю цілісності даних
    /// </summary>
    public class FileAnalyzer
    {
        #region Delegates

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

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

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

        #endregion Delegates

        #region Public Properties & Data

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

                } else
                {
                    return false;
                }
            }
        }

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

```
{
    get
    {
        // Якщо клас не зайнятий обробкою - повертаємо значення
        if (!InProcessing)
        {
            return this.finished;
        } else
        {
            return false;
        }
    }
}
/// <summary>
/// Екземпляр класу повністю закінчив обробку?
/// </summary>
private bool finished;
/// <summary>
/// Булевська властивість "Множина файлів оброблена коректно?"
/// </summary>
public bool ProcessedOK
{
    get
    {
        // Якщо клас не зайнятий обробкою - повертаємо значення
        if (!InProcessing)
        {
            return this.processedOK;
        } else
        {
            return false;
        }
    }
}
/// <summary>
/// Обробка набору файлів зроблена коректно?
/// </summary>
private bool processedOK;

/// <summary>
/// Список порядкових номерів наявних томів
/// </summary>
public int[] VolList
{
    get
    {
        if (!InProcessing)
        {
            return this.volList;
        } else
        {
            return null;
        }
    }
}
/// <summary>
/// Вектор, що вказує на состав томів
/// </summary>
private int[] volList;
/// <summary>
/// Всі томи для відновлення коректні?
/// </summary>
public bool AllEccVolsOK
{
    get
    {
        if (!InProcessing)
        {
            return this.allEccVolsOK;
```

```
                } else
                {
                    return false;
                }
            }
        }
        /// <summary>
        /// Всі томи для відновлення коректні?
        /// </summary>
        private bool allEccVolsOK;
        /// <summary>
        /// Пріоритет процесу
        /// </summary>
        public int ThreadPriority
        {
            get
            {
                return (int)this.threadPriority;
            }
            set
            {
                if (
                        (this.thrFileAnalyzer != null)
                     &&
                        (this.thrFileAnalyzer.IsAlive)
                    )
                {
                    switch (value)
                    {
                        default:
                        case 0:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Lowest;
                            break;
                        }
                        case 1:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.BelowNormal;
                            break;
                        }
                        case 2:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Normal;
                            break;
                        }
                        case 3:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.AboveNormal;
                            break;
                        }
                        case 4:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Highest;
                            break;
                        }
                    }
                    // Встановлюю обраний пріоритет процесу
                    this.thrFileAnalyzer.Priority = this.threadPriority;
                    //Дублюєм встановлення 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, встановлювана за завершенням обробки
/// </summary>
public ManualResetEvent[] FinishedEvent
{
    get
    {
        return this.finishedEvent;
    }
}
/// <summary>
/// Подія, встановлювана за завершенням обробки
/// </summary>
private ManualResetEvent[] finishedEvent;
#endregion Public Properties & Data
#region Data
/// <summary>
/// Модуль для впакування (розпакування) ім'я файлу в префіксний формат
/// </summary>
private FileNamer eFileNamer;
/// <summary>
/// Екземпляр класу контролю цілісності набору файлів
/// </summary>
private FileIntegrityCheck eFileIntegrityCheck;
/// <summary>
/// Шлях до файлів для обробки
/// </summary>
private String path;
/// <summary>
/// Ім'я файлу, якому належить безліч томів
/// </summary>
private String fileName;
/// <summary>
/// Кількість основних томів
/// </summary>
private int dataCount;
/// <summary>
/// Кількість томів для відновлення
/// </summary>
private int eccCount;
/// <summary>
/// Тип кодека Ріда-Соломона (за типом матриці кодування, що вико-ся)
/// </summary>
private int codecType;
/// <summary>
/// Використовується швидке добування з томів (без перевірки CRC-64)?
/// </summary>
private bool fastExtraction;
/// <summary>
/// Потік контролю цілісності файлу
/// </summary>
private Thread thrFileAnalyzer;
/// <summary>
/// Подія припинення обробки файлів
/// </summary>
private ManualResetEvent[] exitEvent;
/// <summary>
/// Подія продовження обробки файлів
/// </summary>
private ManualResetEvent[] executeEvent;
/// <summary>
/// Подія "пробудження" циклу очікування
/// </summary>
```

```
        private ManualResetEvent[] wakeUpEvent;
        #endregion Data
        #region Construction & Destruction
        /// <summary>
        /// Конструктор класу перевірки цілісності набору файлів
        /// </summary>
        public FileAnalyzer()
        {
        // Модуль для впакування (розпакування) ім'я файлу в префіксний формат
            this.eFileNamer = new FileNamer();
            // Створюємо екземпляр класу контролю цілісності набору файлів
            this.eFileIntegrityCheck = new FileIntegrityCheck();
            // Шлях до файлів для обробки за замовчуванням порожній
            this.path = "";
            // Ініціалізуємо ім'я файлу за замовчуванням
            this.fileName = "NONAME";
            // Спочатку всі томи для відновлення вважаємо ушкодженими
            this.allEccVolsOK = false;

            // Екземпляр класу повністю закінчив обробку?
            this.finished = true;
            // Обробка зроблена коректно?
            this.processedOK = false;
            // За замовчуванням встановлюється фоновий пріоритет
            this.threadPriority = 0;
            // Ініціалізуємо подію припинення обробки файлів
            this.exitEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

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

```
            }
            // Скидаємо прапор коректності результату перед запуском потоку
            this.processedOK = false;
            // Скидаємо індикатор актуального стану змінних-членів
            this.finished = false;
            // Зберігаємо шлях до файлів для обробки
            if (path == null)
            {
                this.path = "";
            } else
            {
                // Робимо виділення шляху з "path" у випадку,
                // якщо туди було записано повне ім'я
                this.path = this.eFileNamer.GetPath(path);
            }
            if (fileName == null)
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;
                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();
                return false;
            }
            // Робимо виділення короткого ім'я файлу з "fileName" у випадку,
            // якщо туди було записано повне ім'я
            this.fileName = this.eFileNamer.GetShortFileName(fileName);

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

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

                return false;
            }
            // Зберігаємо кількість основних томів
            this.dataCount = dataCount;
            // Зберігаємо кількість томів для відновлення
            this.eccCount = eccCount;
            // Зберігаємо тип кодека Ріда-Соломона (по типу використовуваної
матриці кодування)
            this.codecType = codecType;
            // Указуємо, що потік повинен виконуватися
            this.exitEvent[0].Reset();
            this.executeEvent[0].Set();
            this.wakeUpEvent[0].Reset();
            this.finishedEvent[0].Reset();
            // Якщо зазначено, що не потрібен запуск в окремому потоці,
            // запускаємо в даному
            if (!runAsSeparateThread)
            {
                // Обчислюємо CRC-64 для кожного з файлів набору
                WriteCRC64();
                // Повертаємо результат обробки
                return this.processedOK;
            }
            // Створюємо потік обчислення й запису CRC-64...
            this.thrFileAnalyzer = new Thread(new ThreadStart(WriteCRC64));
            //...потім даємо йому ім'я...
            this.thrFileAnalyzer.Name = "FileAnalyzer.WriteCRC64()";
```

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

            //...і запускаємо його
            this.thrFileAnalyzer.Start();
            // Повідомляємо, що все нормально
            return true;
        }

        /// <summary>
        /// Метод запуску потоку обробки перевірки CRC64, записаного в кінець
        /// кожного з файлів набору, з генеруванням списку наявних томів
"volList",
        /// який буде використаний декодером для відновлення даних
        /// </summary>
        /// <param name="path">Шлях до файлів для обробки</param>
        /// <param name="fileName">Ім'я файлу для обробки</param>
        /// <param name="dataCount">Конфігурація кількості основних
томів</param>
        /// <param name="eccCount">Конфігурація кількості томів для
відновлення</param>
        /// <param name="codecType">Тип кодека Ріда-Соломона (по типу
матриці)</param>
        /// <param name="fastExtraction">Використовується швидке добування з
томів (без перевірки CRC-64)?</param>
        /// <param name="runAsSeparateThread">Запускати в окремому
потоці?</param>
        /// <returns>Булевський прапор операції</returns>
        public bool StartToAnalyzeCRC64(String path, String fileName, int
dataCount, int eccCount, int codecType, bool fastExtraction, bool
runAsSeparateThread)
        {
            // Якщо потік обчислення CRC-64 працює - не дозволяємо повторний
запуск
            if (InProcessing)
            {
                return false;
            }
            // Спочатку всі томи для відновлення вважаємо ушкодженими
            this.allEccVolsOK = false;
            // Скидаємо прапор коректності результату перед запуском потоку
            this.processedOK = false;
            // Скидаємо індикатор актуального стану змінних-членів
            this.finished = false;
            // Зберігаємо шлях до файлів для обробки
            if (path == null)
            {
                this.path = "";
            } else
            {
                // Робимо виділення шляху з "path" у випадку,
                // якщо туди було записано повне ім'я
                this.path = this.eFileNamer.GetPath(path);
            }

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

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();
                return false;
            }
            // Робимо виділення короткого ім'я файлу з "fileName" у випадку,
            // якщо туди було записано повне ім'я
            this.fileName = this.eFileNamer.GetShortFileName(fileName);

            // Перевіряємо на некоректну конфігурацію
```

```
            if (
                    (dataCount <= 0)
                 ||
                    (eccCount <= 0)
                 ||
                    ((dataCount + eccCount) > (int)RSConst.MaxVolCountAlt)
                )
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

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

                return false;
            }

            // Зберігаємо кількість основних томів
            this.dataCount = dataCount;
            // Зберігаємо кількість томів для відновлення
            this.eccCount = eccCount;
            // Зберігаємо тип кодека Ріда-Соломона (по типу використовуваної
матриці кодування)
            this.codecType = codecType;
            // Використовується швидке добування з томів (без перевірки CRC-64)?
            this.fastExtraction = fastExtraction;

            // Указуємо, що потік повинен виконуватися
            this.exitEvent[0].Reset();
            this.executeEvent[0].Set();
            this.wakeUpEvent[0].Reset();
            this.finishedEvent[0].Reset();
            // Якщо зазначено, що не потрібен запуск в окремому потоці,
            // запускаємо в даному
            if (!runAsSeparateThread)
            {
                // Обчислюємо й перевіряємо CRC-64 для кожного з файлів набору
із заповненням
                // властивості VolList
                AnalyzeCRC64();

                // Повертаємо результат обробки
                return this.processedOK;
            }
            // Створюємо потік обчислення й перевірки CRC-64...
            this.thrFileAnalyzer = new Thread(new ThreadStart(AnalyzeCRC64));

            //...потім даємо йому ім'я...
            this.thrFileAnalyzer.Name = "FileAnalyzer.AnalyzeCRC64()";
            //...установлюємо обраний пріоритет завдання...
            this.thrFileAnalyzer.Priority = this.threadPriority;
            //...і запускаємо його
            this.thrFileAnalyzer.Start();
            // Повідомляємо, що все нормально
            return true;
        }
        /// <summary>
        /// Метод зупинки потоку
        /// </summary>
        public void Stop()
        {
            // Указуємо, що потік обробки більше не повинен виконуватися
            this.exitEvent[0].Set();
            // Примусово знімаємо з паузи
            this.executeEvent[0].Set();

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

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

            // Знімаємо з очікування в циклі
            this.wakeUpEvent[0].Set();
        }
        /// <summary>
        /// Зняття потоку обробки з паузи
        /// </summary>
        public void Continue()
        {
            // Знімаємо обробку з 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",
                        // 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
                // установлена їм Булевська властивість, можливо, ще
                // "не виявилося"
                for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                {
                    if (!this.eFileIntegrityCheck.Finished)
                    {
                        Thread.Sleep((int)WaitTime.MinWaitTime);
```

```
                    } else
                    {
                        break;
                    }
                }
        // Якщо цикли очікування закриття файлових потоків не привели до бажаного
                // результату - це помилка
                if (!this.eFileIntegrityCheck.ProcessedOK)
                {
                    // Указуємо на те, що обробка не була завершена коректно
                    this.processedOK = false;
                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;
                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return;
                }
                // Виводимо прогрес обробки
                if (
                        ((volNum % progressMod1) == 0)
                     &&
                        (OnUpdateFileAnalyzeProgress != null)
                    )
                {
                    OnUpdateFileAnalyzeProgress(((double)(volNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                }
                // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                // буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);
                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Указуємо на те, що обробка була перервана
                    this.processedOK = false;
                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;
                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return;
                }
            }
            // Повідомляємо про закінчення процесу обробки
            if (OnFileAnalyzeFinish != null)
            {
                OnFileAnalyzeFinish();
            }

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

            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;
            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }
          /// <summary>
        /// Обчислення й перевірка значення CRC-64, записаного наприкінці файлу
        /// </summary>
        private void AnalyzeCRC64()
        {
            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = (this.dataCount + this.eccCount) / 100;

           // Якщо модуль дорівнює нулю, то збільшуємо його до значення "1", щоб
            // прогрес виводився на кожній ітерації (файл дуже маленький)
```

```
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }

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

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

            // Індекс у масиві томів
            int volListIdx = 0;

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

            // Лічильник кількості ушкоджених основних томів
            int dataVolMissCount = 0;

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

            // Ім'я файлу для обробки
            String fileName;

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

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

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

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

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

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

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

                                //...якщо одержали сигнал до того, щоб прокинутися -
                            // переходимо на нову ітерацію, тому що прокидаємося
                                // перед постановкою на паузу...
                                if (eventIdx == 0)
                                {
                        //...попередньо скинувши подію, що змусила нас прокинутися
                                    this.wakeUpEvent[0].Reset();
                                    continue;
                                }
                                //...якщо одержали сигнал до виходу з обробки...
                                if (eventIdx == 1)
                                {
                                    //...зупиняємо контрольований алгоритм
                                    this.eFileIntegrityCheck.Stop();

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

                               // Активуємо індикатор актуального стану змінних-членів
                                    this.finished = true;
                                    // Установлюємо подію завершення обробки
                                    this.finishedEvent[0].Set();
                                    return;
                                }
        //...якщо одержали сигнал про завершення обробки вкладеним алгоритмом...
                                if (eventIdx == 2)
                                {
                                    //...exitимо із циклу очікування завершення
(цього й чекали в while(true)!)
                                    break;
                                }
                            } // while(true)
                        } else
                        {
                            // Скидаємо прапор коректності результату
                            this.processedOK = false;
                        // Активуємо індикатор актуального стану змінних-членів
                            this.finished = true;
                            // Установлюємо подію завершення обробки
                            this.finishedEvent[0].Set();
                            return;
                        }
                    // У зв'язку із закриттям великої кількості файлових потоків
                        // необхідно дочекатися запису змін, внесених потоком
                        // кодування в тіло класу. Потік уже не працює, але
                        // установлене їм Булевська властивість, можливо, ще
                        // "не виявилося"
                        for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                        {
                            if (!this.eFileIntegrityCheck.Finished)
                            {
                                Thread.Sleep((int)WaitTime.MinWaitTime);

                            } else
                            {
                                break;
                            }
                        }
                        // Вказуємо, що основний том коректний
                        if (this.eFileIntegrityCheck.ProcessedOK)
                        {
                            dataVolIsOK = true;
                        }
                    } else
```

```
                    {
                        // Вказуємо, 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
                    {
                        // Вказуємо, 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;

            // Встановлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }
        #endregion Private Operations
    }
}
```

Файл RSRaidSSDBase.cs – робота з RAID масивами

```
using System;
using System.Threading;

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

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

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

        #endregion Delegates

        #region Public Properties & Data

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

                } else
                {
                    return false;
                }
            }
        }

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

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

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

    /// <summary>
    /// Булева властивість "Екземпляр класу закінчив обробку
    /// (має актуальний стан змінних-членів)?"
    /// </summary>
    public bool Finished
    {
        get
        {
            // Якщо клас не зайнятий обробкою - повертаємо значення
            if (!InProcessing)
            {
                return this.finished;

            } else
            {
                return false;
            }
        }
    }

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

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

            } else
            {
                return -1;
            }
        }
    }
    /// <summary>
    /// Кількість основних томів
    /// </summary>
    protected int n;
    /// <summary>
    /// Кількість томів для відновлення
    /// </summary>
    public int EccCount
    {
        get
        {
            if (!InProcessing)
            {
                return this.m;
```

```
                } else
                {
                    return -1;
                }
            }
        }
        /// <summary>
        /// Кількість томів для відновлення
        /// </summary>
        protected int m;
        /// <summary>
        /// Тип кодека (за типом використовуваної матриці)
        /// </summary>
        public int CodecType
        {
            get
            {
                if (!InProcessing)
                {
                    return this.eRSType;
                } else
                {
                    return -1;
                }
            }
        }
        /// <summary>
        /// Тип кодека Ріда-Соломона (за типом використовуваної матриці
кодування)
        /// </summary>
        protected int eRSType;

        /// <summary>
        /// Пріоритет процесу
        /// </summary>
        public int ThreadPriority
        {
            get
            {
                return (int)this.threadPriority;
            }
            set
            {
                if (
                        (this.thrRSMatrixForming != null)
                    &&
                        (this.thrRSMatrixForming.IsAlive)
                   )
                {
                    switch (value)
                    {
                        default:
                        case 0:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Lowest;
                            break;
                        }
                        case 1:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.BelowNormal;
                            break;
                        }
                        case 2:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Normal;
```

```
                            break;
                        }
                        case 3:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.AboveNormal;
                            break;
                        }
                        case 4:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Highest;
                            break;
                        }
                    }
                    // Установлюємо обраний пріоритет процесу
                    this.thrRSMatrixForming.Priority = this.threadPriority;
                }
            }
        }
        /// <summary>
        /// Пріоритет процесу підготовки матриці кодування
        /// </summary>
        protected ThreadPriority threadPriority;
        /// <summary>
        /// Подія, що виконуєтсься по завершенню обробки
        /// </summary>
        public ManualResetEvent[] FinishedEvent
        {
            get
            {
                return this.finishedEvent;
            }
        }
        /// <summary>
        /// Подія, що виконується по завершенню обробки
        /// </summary>
        protected ManualResetEvent[] finishedEvent;
        #endregion Public Properties & Data
        #region Data
        // Об'єкт класу роботи з елементами поля Галуа
        protected GF16 eGF16;
        // Матриця RAID-подібного кодера Ріда-Соломона
        protected int[] FLog;
        // Дисперсна матриця (використовується замість "A")
        protected int[] D;
        // "Альтернативна" матриця (використовується замість "D")
        protected int[] A;
        // Основна конфігурація змінилася?
        protected bool mainConfigChanged;
        // Кількість ітерацій першої й другої стадій підготовки матриці
кодування
        protected double iterOfFirstStage;
        protected double iterOfSecondStage;
        // Потік заповнення матриці "FLog" перед виконанням кодування /
декодування
        protected Thread thrRSMatrixForming;
        // Подія припинення підготовки матриці кодування
        protected ManualResetEvent[] exitEvent;
        // Подія продовження підготовки матриці кодування
        protected ManualResetEvent[] executeEvent;
        #endregion Data
        #region Construction & Destruction
        /// <summary>
        /// Конструктор базового класу сутності "RAID-подібний кодек Ріда-
Соломона"
        /// </summary>
        public RSRaidSSDBase()
        {
```

```
            // Створюємо екземпляр класу для роботи з арифметикою поля Галуа (2^16)
            this.eGF16 = new GF16();
            // Екземпляр класу повністю закінчив обробку?
            this.finished = true;
            // Основна конфігурація змінилася?
            this.mainConfigChanged = true;
            // Екземпляр класу ініціалізовано коректно (придатний до роботи)?
            this.configIsOK = false;
            // По-замовчанню встановлюється фоновий пріоритет
            this.threadPriority = 0;
            // Ініціалізуємо подію припинення обробки файлу
            this.exitEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };
            // Ініціалізуємо подію продовження обробки файлу
            this.executeEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

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

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Запуск процесу заповнення матриці "FLog" даними
        /// </summary>
        /// <param name="runAsSeparateThread">Запускати в окремому
потоці?</param>
        /// <returns>Булевий прапор операції</returns>
        public bool Prepare(bool runAsSeparateThread)
        {
            // Якщо потік формування матриці "FLog" працює - не дозволяємо
повторний запуск
            if (InProcessing)
            {
                return false;
            }
            // Якщо конфігурація встановлена некоректно - виходимо
            if (!this.configIsOK)
            {
                return false;
            }
            // Скидаємо індикатор актуального стану змінних-членів
            this.finished = false;
            // Скидаємо подію завершення обробки
            this.finishedEvent[0].Reset();
            // Вказуємо, що потік повинен виконуватися
            this.exitEvent[0].Reset();
            this.executeEvent[0].Set();
            // Якщо зазначено, що не потрібен запуск в окремому потоці,
            // запускаємо в даному
            if (!runAsSeparateThread)
            {
                // Заповнюємо матрицю кодування
                FillFLog();
                // Повертаємо результат обробки
                return this.configIsOK;
            }
            // Створюємо потік формування матриці "FLog"...
            this.thrRSMatrixForming = new Thread(new ThreadStart(FillFLog));
            //...потім даємо йому ім'я...
            this.thrRSMatrixForming.Name = "RSRaidSSD.FillFLog()";
            //...встановлюємо обраний пріоритет завдання...
            this.thrRSMatrixForming.Priority = this.threadPriority;
            //...і запускаємо
            this.thrRSMatrixForming.Start();
```

```
            return true;
        }
        /// <summary>
        /// Метод зупинки потоку
        /// </summary>
        public void Stop()
        {
            // Вказуємо, що потік обробки більше не повинен виконуватися
            this.exitEvent[0].Set();
            // Примусово знімаємо з паузи
            this.executeEvent[0].Set();
        }
        /// <summary>
        /// Постановка потоку обробки на паузу
        /// </summary>
        public void Pause()
        {
            // Ставимо на паузу
            this.executeEvent[0].Reset();
        }
        /// <summary>
        /// Зняття потоку обробки з паузи
        /// </summary>
        public void Continue()
        {
            // Знімаємо обробку з паузи
            this.executeEvent[0].Set();
        }
        #endregion Public Operations
        #region Protected Operations

        /// <summary>
        /// Нормалізація значень "n" і "m" з метою запобігання переповнення
змінних,
        /// ітерацій, що зберігають загальну кількість
        /// </summary>
        protected void NormalizeNM(ref double n, ref double m)
        {
            double maxVal = 0;
            if (n > m)
            {
                maxVal = n;
            } else
            {
                maxVal = m;
            }
            double divider = maxVal / 100.0;
            if (divider > 1)
            {
                n /= divider;
                m /= divider;
            }
        }

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

```
            }
            return -1;
        }
        /// <summary>
        /// Метод перестановки двох рядків місцями
        /// </summary>
        /// <param name="rowNum1">Індекс першого рядка</param>
        /// <param name="rowNum2">Індекс другого рядка</param>
        protected void SwapRows(int rowNum1, int rowNum2)
        {
            // Обчислюємо зсув до елементів i-ой рядка
            int rowNum1this_n = rowNum1 * this.n;
            int rowNum2this_n = rowNum2 * this.n;
            for (int j = 0; j < this.n; j++)
            {
                int dIdx1 = rowNum1this_n + j;
                int dIdx2 = rowNum2this_n + j;
                int tmp       = this.D[dIdx1];
                this.D[dIdx1] = this.D[dIdx2];
                this.D[dIdx2] = tmp;
            }
        }
        /// <summary>
        /// Метод одержання дисперсної матриці "D"
        /// </summary>
        /// <returns>Булевий прапор результату операції</returns>
        protected bool MakeDispersalMatrix()
        {
            // Виділяємо пам'ять під матрицю "FLog"
            this.D = new int[(this.n + this.m) * this.n];
            // Заповнюємо матрицю даними (формуємо матрицю Вандермонда)
            for (int i = 0; i < (this.n + this.m); i++)
            {
                // Зсув у масиві до елементів i-ой рядка
                int i_n = i * this.n;

                // Обчислення рядка матриці Вандермонда (цей блок обчислень
                // може бути реалізований і без використання функції зведення
                // елемента в ступінь, але поточна реалізація припускає більшу
                // гнучкість і зрозумілість)
                for (int j = 0; j < this.n; j++)
                {
                    this.D[i_n + j] = this.eGF16.Pow(i, j);
                }
            }
            // Обчислюємо розподіл відсотків ітерацій по стадіях для
            // коректної обробки відсотків
            double allStageIter  = this.iterOfFirstStage +
this.iterOfSecondStage;
            int percOfFirstStage = (int)((100.0 * this.iterOfFirstStage) /
allStageIter);
            // Дана стадія повинна займати хоча б один відсоток
            // (для коректності розрахунків)
            if (percOfFirstStage == 0)
            {
                percOfFirstStage = 1;
            }
           // Обчислюємо значення модуля, що дозволить виводити відсоток обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = this.n / percOfFirstStage;
           //Якщо модуль дорівнює нулю, то збільшуємо його до значення "1", щоб
            // прогрес виводився на кожній ітерації
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }

            // Цикл вибору діагонального елемента
            for (int k = 1; k < this.n; ++k)
```

```
            {
                // Шукаємо рядок, у якій елемент на головній
                // діагоналі міг би бути ненульовим
                int swapIdx = FindSwapRow(k);
                // Якщо підходящий рядок не може бути знайдений -
                // це помилка - ...
                if (swapIdx == -1)
                {
                    //...вказуємо на помилку конфігурації
                    this.configIsOK = false;
                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;
                    // Встановлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return false;
                }
                // Якщо був знайдений рядок, відмінна від поточної...
                if (swapIdx != k)
                {
                    //...міняємо рядки місцями
                    SwapRows(swapIdx, k);
                }
                int k_n = k * this.n;

                // Витягаємо діагональний елемент
                int diagElem = this.D[k_n + k];
            // Якщо діагональний елемент не дорівнює "1", множимо весь стовпець
                // на зворотний йому елемент, перетворюючи діагональний в "1"
                if (diagElem != 1)
                {
                    // Обчислюємо зворотний елемент для "diagElem"
                    int diagElemInv = this.eGF16.Inv(diagElem);
                    // Робимо необхідну обробку елементів стовпця -
                    // множимо його на елемент, зворотний "diagElem"
                    for (int i = k; i < (this.n + this.m); i++)
                    {
                        int dIdx = (i * this.n) + k;
                        this.D[dIdx] = this.eGF16.Mul(this.D[dIdx],
diagElemInv);
                    }
                }

                // Для всіх стовпців...
                for (int j = 0; j < this.n; j++)
                {
                    // Витягаємо множник поточного стовпця
                    int colMult = this.D[k_n + j];
                    //...не є стовпцями розв'язного елемента...
                    if (
                            (j != k)
                         &&
                            (colMult != 0)
                        )
                    {
                        for (int i = k; i < (this.n + this.m); i++)
                        {
                            int i_n  = i * this.n;
                            int dIdx = i_n + j;
                            //...робимо заміну Cj = Cj - Dk,j * Ck
                            this.D[dIdx] = this.D[dIdx] ^
this.eGF16.Mul(colMult, this.D[i_n + k]);
                        }
                    }
                }
                // Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((k % progressMod1) == 0)
                     &&
                        (OnUpdateRSMatrixFormingProgress != null)
```

```
                    )
                {
                    //...виводимо дані
                    OnUpdateRSMatrixFormingProgress(((double)(k + 1) /
(double)this.n) * percOfFirstStage);
                }
                // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                // буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);
                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Вказуємо, що декодер не сконфігурований коректно
                    this.configIsOK = false;
                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;
                    // Встановлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return false;
                }
            }
            return true;
        }
        /// <summary>
        /// Метод одержання "альтернативної" матриці "A" : у ньому для
заповнення матриці кодування
        /// з 65535 констант вибираються 32768, таких, щоб логарифм кожної з них
був взаємно
        /// простим зі значенням "65535", тобто щоб їх НСД (найбільший спільний
дільник) був рівний
        /// "1". Порушення цієї умови приводить до неможливості обігу матриці
кодування,
        /// і, відповідно, до неможливості відновлення даних)
        /// </summary>
        /// <returns>Булевий прапор результату операції</returns>
        protected bool MakeAlternativeMatrix()
        {
            // Перебирається значення логарифму з метою подальшого одержання
константи
            // для занесення в матрицю шляхом його потенцирування
            int logBase = 0;
            // Відновлення по "logBase" підстановка ступеня для формування рядка
            // матриці Вандермонда
            int powBase = 0;

            // Виділяємо пам'ять під матрицю "FLog"
            this.A = new int[this.m * this.n];
            // Обчислюємо розподіл відсотків ітерацій по стадіях для
            // коректної обробки відсотків
            double allStageIter  = this.iterOfFirstStage +
this.iterOfSecondStage;
            int percOfFirstStage = (int)((100.0 * this.iterOfFirstStage) /
allStageIter);
            // Дана стадія повинна займати хоча б один відсоток
            // (для коректності розрахунків)
            if (percOfFirstStage == 0)
            {
                percOfFirstStage = 1;
            }
           //Обчислюємо значення модуля, що дозволить виводити відсоток обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = this.m / percOfFirstStage;
            //Якщо модуль дорівнює нулю, то збільшуємо його до значення "1", щоб
            // прогрес виводився на кожній ітерації
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }
```

```
            // Заповнюємо матрицю даними (формуємо матрицю Вандермонда)
            for (int i = 0; i < this.m; i++)
            {
                // Поки "logBase" не взаємно просто з "65535"...
                while (
                            ((logBase % 3)   == 0)
                        ||
                            ((logBase % 5)   == 0)
                        ||
                            ((logBase % 17)  == 0)
                        ||
                            ((logBase % 257) == 0)
                      )
                {
                    ++logBase;
                }
                //...потім, відновлюємо його значення...
                powBase = this.eGF16.Exp(logBase++);
                // Зсув у масиві до елементів i-ой рядка
                int i_n = i * this.n;
                for (int j = 0; j < this.n; j++)
                {
                    //...і використовуємо для формування рядка матриці
Вандермонда
                    this.A[i_n + j] = this.eGF16.Pow(powBase, j);
                }
                // Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((i % progressMod1) == 0)
                     &&
                        (OnUpdateRSMatrixFormingProgress != null)
                   )
                {
                    //...виводимо дані
                    OnUpdateRSMatrixFormingProgress(((double)(i + 1) /
(double)this.m) * percOfFirstStage);
                }
                // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                // буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);
                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Вказуємо, що декодер не сконфігуровано коректно
                    this.configIsOK = false;

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

                    // Встановлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return false;
                }
            }
            return true;
        }
        /// <summary>
        /// Заповнення матриці "FLog" даними
        /// </summary>
        protected virtual void FillFLog() { }

        #endregion Protected Operations
    }
}
```

Файл RSRaidSSDEncoder.cs - кодування алгоритмом Ріда-Соломона

```
using System;
using System.Threading;

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

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

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

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

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

```
            int maxVolCount;

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

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

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

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

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

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

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

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

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

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

```
        /// <summary>
        /// Метод множення матриці кодування на вхідний прологарифмований вектор
        /// </summary>
        /// <param name="dataLog">Прологарифмований вхідний вектор (вихідні
дані)</param>
        /// <param name="ecc">Вихідний вектор (надлишкові дані)</param>
        /// <returns>Булевський прапор результату операції</returns>
        public bool Process(int[] dataLog, ref int[] ecc)
        {
            // Якщо кодер зконфігуровано некоректно, обробка неможлива!
            if (!this.configIsOK)
            {
                return false;
            }
            // Копіюємо покажчик на масив експонент для скорочення часу обігу
            int[] GF16Exp = this.eGF16.GFExpTable;
            // Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.m; i++)
            {
                int mulSum = 0;       // Сума добутку рядка матриці на стовпець
                int i_n = i * this.n;   // Зсув у масиві до елементів i-ой рядка
                for (int j = 0; j < this.n; j++)
                {
                    mulSum ^= GF16Exp[this.FLog[i_n + j] + dataLog[j]];
                }
                ecc[i] = mulSum;
            }
            return true;
        }
        #endregion Public Operations
        #region Private Operations
        /// <summary>
        /// Заповнення матриці Вандермонда даними
        /// </summary>
        protected override void FillFLog()
        {
            // Якщо основна конфігурація змінилася...
            if (this.mainConfigChanged)
            {
                if (this.eRSType == (int)RSType.Dispersal)
                {
                    //...робимо формування дисперсної матриці "D"
                    if (!MakeDispersalMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                } else
                {
                   //...робимо формування альтернативного заповнення матриці "A"
                    if (!MakeAlternativeMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                }
                // Виділяємо пам'ять під матрицю "FLog"
                this.FLog = new int[this.m * this.n];
                // Заповнюємо матрицю кодування
```

```
                for (int i = 0; i < this.m; i++)
                {
                    // Зсув у масиві до елементів i-ой рядка
                    int i_n = i * this.n;
                  // Залежно від типу декодера беремо дані з відповідного масиву
                    if (this.eRSType == (int)RSType.Dispersal)
                    {
                        // Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
                // У матрицю кодування поміщаємо логарифми її вихідних елементів
                            // (для прискорення множення матриці на вектор)
                            this.FLog[i_n + j] = this.eGF16.Log(this.D[((this.n
+ i) * this.n) + j]);
                        }
                    } else
                    {
                        // Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
                            int idx = i_n + j;
               // У матрицю кодування поміщаємо логарифми її вихідних елементів
                            // (для прискорення множення матриці на вектор)
                            this.FLog[idx] = this.eGF16.Log(this.A[idx]);
                        }
                    }
                    // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                    // буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);
                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                }

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

            // Якщо є передплата на делегата завершення...
            if (OnRSMatrixFormingFinish != null)
            {
                //...повідомляємо, що екземпляр класу готовий до роботи
                OnRSMatrixFormingFinish();
            }
            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;
            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }
        #endregion Private Operations
    }
}
```

Файл RSRaidSSDDecoder.cs - декодування алгоритмомом Ріда-Соломона

```
using System;
using System.Threading;

namespace RecoveryDisk
{
    /// <summary>
    /// Клас RAID-подібного декодера Ріда-Соломона
    /// </summary>
    public class RSRaidSSDDecoder : RSRaidSSDBase
    {
        #region Data

        // Масив булевських ознак "рядок матриці "FLog" тривіальна?"
        private bool[] FLogRowIsTrivial;

        // Список порядкових номерів наявних томів (нумерація з нуля)
        private int[] volList;

        #endregion Data

        #region Construction & Destruction

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

        /// <summary>
        /// Базовий конструктор декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        public RSRaidSSDDecoder(int dataCount, int eccCount, int[] volList)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList, (int)RSType.Alternative);

            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }
        /// <summary>
        /// Розширений конструктор декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        /// <param name="codecType">Тип кодека Ріда-Соломона (по типу
матриці)</param>
        public RSRaidSSDDecoder(int dataCount, int eccCount, int[] volList, int
codecType)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList, codecType);

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

```
        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Установка конфігурації декодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="volList">Список порядкових номерів наявних
томів</param>
        /// <param name="codecType">Тип кодека Ріда-Соломона (по типу
матриці)</param>
        /// <returns>Булевський прапор операції установки конфігурації</returns>
        public bool SetConfig(int dataCount, int eccCount, int[] volList, int
codecType)
        {
            int maxVolCount;

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

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

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

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

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

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

         // Кількість ітерацій, що відслідковуються прогресом, на першій стадії
                // залежить від типу використовуваної матриці
                if (this.eRSType == (int)RSType.Alternative)
                {
                    this.iterOfFirstStage = m;
```

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

                this.iterOfSecondStage = (n * (((n - 1) * (n - 1)) + (n * n)));

                // Виділяємо пам'ять під масив булевських ознак "рядок матриці
"FLog" тривіальна?"
                this.FLogRowIsTrivial = new bool[dataCount];

                // Зберігаємо список наявних томів
                this.volList = volList;

                this.configIsOK = true;

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

            return this.configIsOK;
        }

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

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

            // Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.n; i++)
            {
                // Якщо поточний рядок матриці не є тривіальної, робимо обробку
                if (!this.FLogRowIsTrivial[i])
                {
                   int mulSum = 0;// Сума добутку рядка матриці на стовпець
                   int i_n = i * this.n;// Зсув у масиві до елементів i-ой рядка
                   for (int j = 0; j < this.n; j++)
                   {
                        mulSum ^= GF16Exp[this.FLog[i_n + j] + dataEccLog[j]];
                    }

                    data[i] = mulSum;

                } else
                {
                    data[i] = GF16Exp[dataEccLog[i]];
                }
            }

            return true;
        }
```

```
        #endregion Public Operations

        #region Private Operations

        /// <summary>
        /// Пошук матриці, зворотної до "FLog", методом Жорданових виключень
        /// (Дана модифікація методу може використовуватися тільки в тих
випадках,
        /// коли (-a) = (a), тому що через непотрібність пропущена стадія зміни
елементів),
        /// крім того, відсутній пошук ненульового розв'язного елемента (у
випадку
        /// роботи з матрицею Вандермонда наявність нуля на діагоналі - збій
кодека,
        /// тому ситуація з виявленням нуля сприймається винятково як помилка
        /// </summary>
        /// <returns>Булевський прапор результату операції</returns>
        private bool FInv()
        {
            // Обчислюємо розподіл відсотків ітерацій по стадіях для
            // коректної обробки відсотків
            double allStageIter   = this.iterOfFirstStage +
this.iterOfSecondStage;
            int percOfFirstStage  = (int)((100.0 * this.iterOfFirstStage)  /
allStageIter);
            int percOfSecondStage = (int)((100.0 * this.iterOfSecondStage) /
allStageIter);

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

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

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

            // Цикл вибору розв'язного елемента "pivot"
            for (int k = 0; k < this.n; ++k)
            {
             //Якщо даний рядок тривіальна - просто переходимо на нову ітерацію
                if (this.FLogRowIsTrivial[k])
                {
                    continue;
                }

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

                // Індекс розв'язного елемента
                int pivotIdx = k_n + k;

                // Витягаємо розв'язний елемент
                int pivot = this.FLog[pivotIdx];

                // Якщо розв'язний елемент дорівнює нулю - матриця не має
зворотної
                if (pivot == 0)
                {
                    //...указуємо на помилку конфігурації
```

```
                    this.configIsOK = false;

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

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

                    return false;
                }

                // Після добування розв'язного елемента поміщаємо на його місце
"1"
                this.FLog[pivotIdx] = 1;

                // Працюємо з рядками...
                for (int i = 0; i < this.n; i++)
                {
                    // Якщо перебуваємо на рядку розв'язного елемента -
переходимо
                    // на нову ітерацію
                    if (i == k)
                    {
                        continue;
                    }

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

                    // Працюємо зі стовпцями...
                    for (int j = 0; j < this.n; j++)
                    {
                        // Якщо перебуваємо на стовпці розв'язного елемента -
переходимо
                        // на нову ітерацію...
                        if (j == k)
                        {
                            continue;
                        }

                        int idx = i_n + j;

                        //...а інакше робимо необхідні дії над матрицею:
                        // "A[i,j] = A[i,j] * pivot + A[i,k] * A[k,j]"
                        this.FLog[idx] = this.eGF16.Mul(this.FLog[idx], pivot) ^
this.eGF16.Mul(this.FLog[i_n + k], this.FLog[k_n + j]);
                    }
                }

                // Розподіл матриці на розв'язний елемент заміняємо множенням на
зворотний
                int pivotInv = this.eGF16.Inv(pivot);

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

                    for (int j = 0; j < this.n; j++)
                    {
                        int idx = i_n + j;

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

                // Якщо є передплата на делегата відновлення прогресу -...
                if (
```

```
                        ((k % progressMod1) == 0)
                     &&
                        (OnUpdateRSMatrixFormingProgress != null)
                    )
                {
                    //...виводимо дані
                    OnUpdateRSMatrixFormingProgress((((double)(k + 1) /
(double)this.n) * percOfSecondStage) + percOfFirstStage);
                }

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

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Указуємо, що декодер зконфігуровано некоректно
                    this.configIsOK = false;

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

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return false;
                }
            }
            return true;
        }
        /// <summary>
        /// Обчислення логарифмів значень інвертованої матриці
        /// <summary>
        private void LogFCalc()
        {
            // Працюємо з рядками...
            for (int i = 0; i < this.n; i++)
            {
                // Зсув у масиві до елементів і-ой рядка
                int i_n = i * this.n;

                // Працюємо зі стовпцями...
                for (int j = 0; j < this.n; j++)
                {
                    int idx = i_n + j;
                    this.FLog[idx] = this.eGF16.Log(this.FLog[idx]);
                }
            }
        }
        /// <summary>
        /// Заповнення матриці "FLog" (матриці декодера) даними
        /// </summary>
        protected override void FillFLog()
        {
            // Якщо довжина вектора наявних томів менше кількості,
            // необхідного для відновлення...
            if (this.volList.Length < this.n)
            {
                //...указуємо на помилку конфігурації
                this.configIsOK = false;

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

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

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

            // Вектор лічильників всіх томів...
            int[] allVolCount = new int[this.n + this.m];

            //...і вектор ecc-томів для "затикання" пробілів, створених
            // загубленими основними томами
            int[] eccVolToFix = new int[this.m];

            // Лічильник кількості стертих основних томів
            int dataVolMissCount = this.n;

            // Ініціалізуємо масив лічильників всіх томів
            for (int i = 0; i < (this.n + this.m); i++)
            {
                allVolCount[i] = 0;
            }

        //Проводимо аналіз складу представлених томів на предмет наявності основних
            for (int i = 0; i < this.n; i++)
            {
                // Обчислюємо номер поточного тому
                int currVol = Math.Abs(this.volList[i]);

                // Якщо номер тому відповідає припустимому діапазону
                if (currVol < (this.n + this.m))
                {
                    ++allVolCount[currVol];

                    // Якщо поточний том є основним, фіксуємо даний факт
                    if (currVol < this.n)
                    {
                        ---idataVolMissCount;
                    }

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

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

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

                    return;
                }
            }

            // Перевіряємо лічильники томів на помилкове дублювання
            for (int i = 0; i < (this.n + this.m); i++)
            {
                // Якщо деякий том був зазначений більш ніж один раз...
                if (allVolCount[i] > 1)
                {
                    //...указуємо на помилку конфігурації
                    this.configIsOK = false;

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

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

                    return;
                }
```

```
            }

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

            // Якщо основна конфігурація змінилася...
            if (this.mainConfigChanged)
            {
                if (this.eRSType == (int)RSType.Dispersal)
                {
                    //...робимо формування дисперсної матриці "D"
                    if (!MakeDispersalMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

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

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                } else
                {
                   //...робимо формування альтернативного заповнення матриці "A"
                    if (!MakeAlternativeMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

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

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                }
                //...і скидаємо прапор
                this.mainConfigChanged = false;
            }

            // Для кожного загубленого основного тому шукаємо том для
відновлення
            for (int i = 0, j = 0; i < dataVolMissCount; i++)
            {
                // Рухаємося за списком томів доти, поки не знайдемо том для
                // відновлення для затикання "дірки" (основні томи мають номера
                // менше this.n (при нумерації з нуля!))
                while (this.volList[j] < this.n)
                {
                    j++;
                }
                // Зберігаємо номер тому для заміни загубленого основного тому
                eccVolToFix[i] = this.volList[j];

                j++; // j++ дозволяє перейти до наступного пошуку
            }

            // Працюємо по рядках матриці (в ідеалі, всі рядки повинні
заповнюватися
            // рядками з одиницею на головній діагоналі, що відповідає
відсутності
            // ушкоджень, але allVolCount укаже, якими є справи з наявністю
томів)
            for (int i = 0, e = 0; i < this.n; i++)
            {
```

```
                // Індекс рядка з дисперсної матриці, що буде поміщена в матрицю
кодування
                int DRowIdx;

                // Зсув у масиві до елементів і-ой рядка
                int i_n = i * this.n;
                // Якщо основний том відсутній, формуємо рядок матриці
Вандермонда
                if (allVolCount[i] == 0)
                {
            // Обчислюємо номер рядка матриці Вандермонда, яку потрібно вставити
                    // на місце даного рядка формованої матриці "FLog"
                    DRowIdx = eccVolToFix[e++];

                    // Указуємо, що даний рядок матриці "FLog" не тривіальна
                    this.FLogRowIsTrivial[i] = false;
                } else
                {
                    // Формуємо в матриці "FLog" нульовий рядок з одиницею на
головній діагоналі
                    // (відповідає наявному основному той)
                    DRowIdx = i;
                    // Указуємо, що даний рядок матриці "FLog" тривіальна
                    this.FLogRowIsTrivial[i] = true;
                }

                // Залежно від типу декодера беремо дані з відповідного масиву
                // (у ньому втримуються як рядки матриці Вандермонда, так і
"тривіальні" рядки,
                //  утримуючі нулі і єдиний елемент "1" на головній діагоналі)
                if (this.eRSType == (int)RSType.Dispersal)
                {
                    int bs = DRowIdx * this.n;
                    // Формування рядка в матриці кодування
                //("тривіальні" рядки вже втримуються в матриці "D", вони вийшли
                    // "автоматично" на попередньому етапі обробки
MakeDispersal())
                    for (int j = 0; j < this.n; j++)
                    {
                        this.FLog[i_n + j] = this.D[bs + j];
                    }
                } else
                {
                    // Якщо це потрібно - формуємо "тривіальний" рядок...
                    if (this.FLogRowIsTrivial[i])
                    {
                        for (int j = 0; j < this.n; j++)
                        {
                            this.FLog[i_n + j] = 0;
                        }
                        this.FLog[i_n + i] = 1;
                    } else
                    {
                        int bs = (DRowIdx - this.n) * this.n;
                        //...а, інакше, беремо рядок матриці Вандермонда
                        for (int j = 0; j < this.n; j++)
                        {
                            this.FLog[i_n + j] = this.A[bs + j];
                        }
                    }
                }

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

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
```

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

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

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

                    return;
                }
            }

            // Знаходимо зворотну матрицю для "FLog"
            if (!FInv())
            {
                // Указуємо, що кодер зконфігуровано некоректно
                this.configIsOK = false;

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

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

                return;
            }

            // Обчислюємо логарифми елементів інвертованої матриці
            LogFCalc();

            // Якщо є передплата на делегата завершення...
            if (OnRSMatrixFormingFinish != null)
            {
                //...повідомляємо, що екземпляр класу готовий до роботи
                OnRSMatrixFormingFinish();
            }
            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;
            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }

        #endregion Private Operations
        #region Public Properties
        /// <summary>
        /// Список порядкових номерів наявних томів
        /// </summary>
        public int[] VolList
        {
            get
            {
                if (!InProcessing)
                {
                    return this.volList;
                } else
                {
                    return null;
                }
            }
        }

        #endregion Public Properties
    }
}
```

Файл MainForm.cs - головне вікно програми

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

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

        #region Windows Form Designer generated code

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

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

```
            this.helpToolStripMenuItem});
            this.menuStrip.LayoutStyle =
System.Windows.Forms.ToolStripLayoutStyle.Flow;
            this.menuStrip.Location = new System.Drawing.Point(0, 0);
            this.menuStrip.Name = "menuStrip";
            this.menuStrip.Size = new System.Drawing.Size(986, 21);
            this.menuStrip.TabIndex = 0;
            this.menuStrip.Text = "menuStrip";
            //
            // fileToolStripMenuItem
            //
            this.fileToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.exitToolStripMenuItem});
            this.fileToolStripMenuItem.Name = "файлToolStripMenuItem";
            this.fileToolStripMenuItem.Size = new System.Drawing.Size(45, 17);
            this.fileToolStripMenuItem.Text = "Файл";
            //
            // instrumentsToolStripMenuItem
            //
            this.instrumentToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.testspeedToolStripMenuItem});
            this.instrumentToolStripMenuItem.Name =
"instrumentsToolStripMenuItem";
            this.instrumentToolStripMenuItem.Size = new System.Drawing.Size(82,
17);
            this.instrumentToolStripMenuItem.Text = "Інструменти";
            //
            // adjustmentToolStripMenuItem
            //
            this.adjustmentToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.encodingfilterToolStripMenuItem,
            this.quickextractionToolStripMenuItem});
            this.adjustmentToolStripMenuItem.Name =
"adjustmentToolStripMenuItem";
            this.adjustmentToolStripMenuItem.Size = new System.Drawing.Size(74,
17);
            this.adjustmentToolStripMenuItem.Text = "Параметри";
            //
            // encodingfilterToolStripMenuItem
            //
            this.encodingfilterToolStripMenuItem.ImageScaling =
System.Windows.Forms.ToolStripItemImageScaling.None;
            this.encodingfilterToolStripMenuItem.Name =
"encodingfilterToolStripMenuItem";
            this.encodingfilterToolStripMenuItem.Size = new
System.Drawing.Size(216, 22);
            this.encodingfilterToolStripMenuItem.Text = "Шифруючий фільтр";
            this.encodingfilterToolStripMenuItem.Click += new
System.EventHandler(this.encodingfilterToolStripMenuItem_Click);
            //
            // helpToolStripMenuItem
            //
            this.helpToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.separatorToolStripMenuItem,
            this.aboutToolStripMenuItem});
            this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
            this.helpToolStripMenuItem.Size = new System.Drawing.Size(60, 17);
            this.helpToolStripMenuItem.Text = "Довідка";
            //
            // separatorToolStripMenuItem
            //
            this.separatorToolStripMenuItem.Name = "separatorToolStripMenuItem";
            this.separatorToolStripMenuItem.Size = new System.Drawing.Size(163,
6);
            //
```

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

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

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

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

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

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

        }

        #endregion

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

Файл ProcessForm.cs - вікно створення та перевірки RaidSSD масивів

```
namespace RecoveryDisk
{
    partial class ProcessForm
    {
        /// <summary>
                /// </summary>
        private System.ComponentModel.IContainer components = null;
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(ProcessForm));
            this.processPriorityGroupBox = new System.Windows.Forms.GroupBox();
            this.processPriorityComboBox = new System.Windows.Forms.ComboBox();
            this.processGroupBox = new System.Windows.Forms.GroupBox();
            this.processProgressBar = new System.Windows.Forms.ProgressBar();
            this.fileAnalyzeStatGroupBox = new System.Windows.Forms.GroupBox();
            this.percOfAltEccLabel = new System.Windows.Forms.Label();
            this.percOfDamageLabel = new System.Windows.Forms.Label();
            this.percOfAltEccLabel_ = new System.Windows.Forms.Label();
            this.percOfDamageLabel_ = new System.Windows.Forms.Label();
            this.logGroupBox = new System.Windows.Forms.GroupBox();
            this.logListBox = new System.Windows.Forms.ListBox();
            this.countGroupBox = new System.Windows.Forms.GroupBox();
            this.errorCountLabel = new System.Windows.Forms.Label();
            this.okCountLabel = new System.Windows.Forms.Label();
            this.errorPictureBox = new System.Windows.Forms.PictureBox();
            this.okPictureBox = new System.Windows.Forms.PictureBox();
            this.errorCountLabel_ = new System.Windows.Forms.Label();
            this.okCountLabel_ = new System.Windows.Forms.Label();
            this.toolTip = new System.Windows.Forms.ToolTip(this.components);
            this.stopButtonXP = new PinkieControls.ButtonXP();
            this.pauseButtonXP = new PinkieControls.ButtonXP();
            this.closingTimer = new System.Windows.Forms.Timer(this.components);
            this.processTimer = new System.Windows.Forms.Timer(this.components);
            this.processPriorityGroupBox.SuspendLayout();
            this.processGroupBox.SuspendLayout();
            this.fileAnalyzeStatGroupBox.SuspendLayout();
            this.logGroupBox.SuspendLayout();
            this.countGroupBox.SuspendLayout();
 ((System.ComponentModel.ISupportInitialize)(this.errorPictureBox)).BeginInit();
 ((System.ComponentModel.ISupportInitialize)(this.okPictureBox)).BeginInit();
            this.SuspendLayout();
            //
            // processPriorityGroupBox
            //
```

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

```
            this.fileAnalyzeStatGroupBox.Location = new System.Drawing.Point(12,
216);
            this.fileAnalyzeStatGroupBox.Name = "fileAnalyzeStatGroupBox";
            this.fileAnalyzeStatGroupBox.Size = new System.Drawing.Size(459,
64);
            this.fileAnalyzeStatGroupBox.TabIndex = 0;
            this.fileAnalyzeStatGroupBox.TabStop = false;
            this.fileAnalyzeStatGroupBox.Text = "Результат аналізу цілісності
даних";
            //
            // percOfAltEccLabel
            //
            this.percOfAltEccLabel.AutoSize = true;
            this.percOfAltEccLabel.Location = new System.Drawing.Point(195, 41);
            this.percOfAltEccLabel.Name = "percOfAltEccLabel";
            this.percOfAltEccLabel.Size = new System.Drawing.Size(10, 13);
            this.percOfAltEccLabel.TabIndex = 0;
            this.percOfAltEccLabel.Text = "-";
            //
            // percOfDamageLabel
            //
            this.percOfDamageLabel.AutoSize = true;
            this.percOfDamageLabel.Location = new System.Drawing.Point(195, 20);
            this.percOfDamageLabel.Name = "percOfDamageLabel";
            this.percOfDamageLabel.Size = new System.Drawing.Size(10, 13);
            this.percOfDamageLabel.TabIndex = 0;
            this.percOfDamageLabel.Text = "-";
            //
            // percOfAltEccLabel_
            //
            this.percOfAltEccLabel_.AutoSize = true;
            this.percOfAltEccLabel_.Location = new System.Drawing.Point(7, 41);
            this.percOfAltEccLabel_.Name = "percOfAltEccLabel_";
            this.percOfAltEccLabel_.Size = new System.Drawing.Size(163, 13);
            this.percOfAltEccLabel_.TabIndex = 0;
            this.percOfAltEccLabel_.Text = "Резерв томів для відновлення:";
            //
            // percOfDamageLabel_
            //
            this.percOfDamageLabel_.AutoSize = true;
            this.percOfDamageLabel_.Location = new System.Drawing.Point(7, 20);
            this.percOfDamageLabel_.Name = "percOfDamageLabel_";
            this.percOfDamageLabel_.Size = new System.Drawing.Size(148, 13);
            this.percOfDamageLabel_.TabIndex = 0;
            this.percOfDamageLabel_.Text = "Всього пошкоджених томів:";
            //
            // logGroupBox
            //
            this.logGroupBox.Controls.Add(this.logListBox);
            this.logGroupBox.Location = new System.Drawing.Point(12, 80);
            this.logGroupBox.Name = "logGroupBox";
            this.logGroupBox.Size = new System.Drawing.Size(871, 130);
            this.logGroupBox.TabIndex = 0;
            this.logGroupBox.TabStop = false;
            this.logGroupBox.Text = "Лог процесу";
            //
            // logListBox
            //
            this.logListBox.BackColor = System.Drawing.SystemColors.Control;
            this.logListBox.BorderStyle = System.Windows.Forms.BorderStyle.None;
            this.logListBox.FormattingEnabled = true;
            this.logListBox.HorizontalScrollbar = true;
            this.logListBox.Location = new System.Drawing.Point(7, 23);
            this.logListBox.Name = "logListBox";
            this.logListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.logListBox.Size = new System.Drawing.Size(851, 91);
            this.logListBox.TabIndex = 0;
            this.logListBox.TabStop = false;
```

```
            this.logListBox.UseTabStops = false;
            this.logListBox.SelectedIndexChanged += new
System.EventHandler(this.logListBox_SelectedIndexChanged);
            //
            // countGroupBox
            //
            this.countGroupBox.Controls.Add(this.errorCountLabel);
            this.countGroupBox.Controls.Add(this.okCountLabel);
            this.countGroupBox.Controls.Add(this.errorPictureBox);
            this.countGroupBox.Controls.Add(this.okPictureBox);
            this.countGroupBox.Controls.Add(this.errorCountLabel_);
            this.countGroupBox.Controls.Add(this.okCountLabel_);
            this.countGroupBox.Location = new System.Drawing.Point(482, 216);
            this.countGroupBox.Name = "countGroupBox";
            this.countGroupBox.Size = new System.Drawing.Size(124, 64);
            this.countGroupBox.TabIndex = 0;
            this.countGroupBox.TabStop = false;
            this.countGroupBox.Text = "Лічільник процесу";
            //
            // errorCountLabel
            //
            this.errorCountLabel.AutoSize = true;
            this.errorCountLabel.Location = new System.Drawing.Point(63, 41);
            this.errorCountLabel.Name = "errorCountLabel";
            this.errorCountLabel.Size = new System.Drawing.Size(13, 13);
            this.errorCountLabel.TabIndex = 0;
            this.errorCountLabel.Text = "0";
            this.toolTip.SetToolTip(this.errorCountLabel, "Лічильник некоректно
оброблених файлів");
            //
            // okCountLabel
            //
            this.okCountLabel.AutoSize = true;
            this.okCountLabel.Location = new System.Drawing.Point(63, 20);
            this.okCountLabel.Name = "okCountLabel";
            this.okCountLabel.Size = new System.Drawing.Size(13, 13);
            this.okCountLabel.TabIndex = 0;
            this.okCountLabel.Text = "0";
            this.toolTip.SetToolTip(this.okCountLabel, "Лічильник коректно
оброблених файлів");
            //
            // errorPictureBox
            //
            this.errorPictureBox.Image =
((System.Drawing.Image)(resources.GetObject("errorPictureBox.Image")));
            this.errorPictureBox.Location = new System.Drawing.Point(10, 40);
            this.errorPictureBox.Name = "errorPictureBox";
            this.errorPictureBox.Size = new System.Drawing.Size(21, 15);
            this.errorPictureBox.TabIndex = 2;
            this.errorPictureBox.TabStop = false;
            //
            // okPictureBox
            //
            this.okPictureBox.Image =
 ((System.Drawing.Image)(resources.GetObject("okPictureBox.Image")));
            this.okPictureBox.Location = new System.Drawing.Point(10, 19);
            this.okPictureBox.Name = "okPictureBox";
            this.okPictureBox.Size = new System.Drawing.Size(21, 15);
            this.okPictureBox.TabIndex = 1;
            this.okPictureBox.TabStop = false;
            //
            // errorCountLabel_
            //
            this.errorCountLabel_.AutoSize = true;
            this.errorCountLabel_.Location = new System.Drawing.Point(28, 41);
            this.errorCountLabel_.Name = "errorCountLabel_";
            this.errorCountLabel_.Size = new System.Drawing.Size(35, 13);
            this.errorCountLabel_.TabIndex = 0;
            this.errorCountLabel_.Text = "Error :";
```

```
            this.toolTip.SetToolTip(this.errorCountLabel_, "Лічильник некоректно
оброблених файлів");
            //
            // okCountLabel_
            //
            this.okCountLabel_.AutoSize = true;
            this.okCountLabel_.Location = new System.Drawing.Point(28, 20);
            this.okCountLabel_.Name = "okCountLabel_";
            this.okCountLabel_.Size = new System.Drawing.Size(28, 13);
            this.okCountLabel_.TabIndex = 0;
            this.okCountLabel_.Text = "OK :";
            this.toolTip.SetToolTip(this.okCountLabel_, "Лічильник коректно
оброблених файлів");
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // stopButtonXP
            //
            this.stopButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.stopButtonXP.DefaultScheme = true;
            this.stopButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.stopButtonXP.Hint = "";
            this.stopButtonXP.Location = new System.Drawing.Point(762, 257);
            this.stopButtonXP.Name = "stopButtonXP";
            this.stopButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.stopButtonXP.Size = new System.Drawing.Size(121, 23);
            this.stopButtonXP.TabIndex = 2;
            this.stopButtonXP.Text = "Перервати обробку";
            this.toolTip.SetToolTip(this.stopButtonXP, "Припинення обробки
файлів із закриттям даного вікна");
            this.stopButtonXP.Click += new
System.EventHandler(this.stopButtonXP_Click);
            //
            // pauseButtonXP
            //
            this.pauseButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.pauseButtonXP.DefaultScheme = true;
            this.pauseButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.pauseButtonXP.Hint = "";
            this.pauseButtonXP.Location = new System.Drawing.Point(762, 220);
            this.pauseButtonXP.Name = "pauseButtonXP";
            this.pauseButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.pauseButtonXP.Size = new System.Drawing.Size(121, 23);
            this.pauseButtonXP.TabIndex = 1;
            this.pauseButtonXP.Text = "Пауза";
            this.toolTip.SetToolTip(this.pauseButtonXP, "Постановка/зняття
процесу обробки з паузи");
            this.pauseButtonXP.Click += new
System.EventHandler(this.pauseButtonXP_Click);
            //
            // closingTimer
            //
            this.closingTimer.Tick += new
System.EventHandler(this.closingTimer_Tick);
            //
            // processTimer
            //
            this.processTimer.Interval = 500;
```

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

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

((System.ComponentModel.ISupportInitialize)(this.okPictureBox)).EndInit();
            this.ResumeLayout(false);
        }
        #endregion
        private System.Windows.Forms.GroupBox processPriorityGroupBox;
        private System.Windows.Forms.GroupBox processGroupBox;
        private System.Windows.Forms.ProgressBar processProgressBar;
        private System.Windows.Forms.GroupBox fileAnalyzeStatGroupBox;
        private System.Windows.Forms.Label percOfDamageLabel_;
        private System.Windows.Forms.Label percOfAltEccLabel_;
        private System.Windows.Forms.GroupBox logGroupBox;
        private System.Windows.Forms.GroupBox countGroupBox;
        private System.Windows.Forms.Label errorCountLabel_;
        private System.Windows.Forms.Label okCountLabel_;
        private System.Windows.Forms.ListBox logListBox;
        private System.Windows.Forms.ComboBox processPriorityComboBox;
        private System.Windows.Forms.PictureBox errorPictureBox;
        private System.Windows.Forms.PictureBox okPictureBox;
        private System.Windows.Forms.Label errorCountLabel;
        private System.Windows.Forms.Label okCountLabel;
        private System.Windows.Forms.ToolTip toolTip;
        private System.Windows.Forms.Timer closingTimer;
        private System.Windows.Forms.Label percOfAltEccLabel;
        private System.Windows.Forms.Label percOfDamageLabel;
        private PinkieControls.ButtonXP pauseButtonXP;
        private PinkieControls.ButtonXP stopButtonXP;
        private System.Windows.Forms.Timer processTimer;
    }
}
```

Файл BenchmarkForm.cs - вікно тестування швидкодії алгоритму

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

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

        #region Windows Form Designer generated code

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

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

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

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

        }

        #endregion

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

Файл About.cs - вікно довідки про програму

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

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

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.developersListBoxdevelopersListBox = new
System.Windows.Forms.ListBox();
            this.RSIconTimer = new System.Windows.Forms.Timer(this.components);
            this.okButtonXP = new PinkieControls.ButtonXP();
            this.SuspendLayout();
            //
            // developersListBoxdevelopersListBox
            //
            this.developersListBoxdevelopersListBox.BackColor =
System.Drawing.SystemColors.Control;
            this.developersListBoxdevelopersListBox.BorderStyle =
System.Windows.Forms.BorderStyle.None;
            this.developersListBoxdevelopersListBox.FormattingEnabled = true;
            this.developersListBoxdevelopersListBox.Items.AddRange(new object[]
{
    "ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ОСВІТИ ",
            "",
            "На тему:",
            "",
            "Дослідження та програмна реалізація системи резервного копіювання",
            " даних хмарного, віртуального й фізичного середовища",
            "",
            "",
            "Керівник проекту: к.ф-м.н., доц. Якименко Н.М.",
            "",
            "Розробив: студентка Вітряченко Аліна Андріївна",
            "                  гр. КІ-21М                                    ",
            "",
            "м. Кропивницький 2022"});
            this.developersListBoxdevelopersListBox.Location = new
System.Drawing.Point(11, 6);
            this.developersListBoxdevelopersListBox.Name =
"developersListBoxdevelopersListBox";
```

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