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
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-22М-2
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Омелян Ю.В.
«____» _____ 2023 р.

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Якименко Н.М.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 122 *"Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерні науки"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2023 року

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

Омеляну Юрію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі*

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

затверджені наказом вищого навчального закладу № 33-13 від 04.08.2023 року

3. Строк подання студентом роботи до захисту *10.12.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі*

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.	05.10.2023	14.11.2023
Охорона праці	Оришака О.В.	06.10.2023	16.11.2023

7. Дата видачі завдання « 6 » вересня 2023 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	10.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2023 р.	
3.	Розробка моделі компонента	20.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2023 р.	
6.	Програмування алгоритмів	10.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2023 р.	
9.	Оформлення ПЗ	17.11.2023 р.	
10.	Попередній захист роботи	10.12.2023 р.	

Дата видачі завдання
« 6 » вересня 2023 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2023 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Омелян Ю.В. Дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи підвищення надійності зберігання даних у хмарі.

Метою розробки є дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі.

Об'єктом дослідження є процес підвищення надійності зберігання даних у хмарі.

Предметом дослідження є методи підвищення надійності зберігання даних у хмарі.

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

Результат роботи – програмна реалізація системи підвищення надійності зберігання даних у хмарі.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

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

Програму розроблено в середовищі Visual C#.

Ключові слова: комп'ютерні науки, надійність, хмара

ABSTRACT

Omelian Yu.V. Research and software implementation of a system for increasing the reliability of data storage in the cloud. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for the system of increasing the reliability of data storage in the cloud.

The purpose of the development is research and software implementation of a system for increasing the reliability of data storage in the cloud.

The object of research is the process of increasing the reliability of data storage in the cloud.

The subject of research is methods of increasing the reliability of data storage in the cloud.

Research methods are based on reliability theory methods, mathematical statistics methods, and software development methods.

The result of the work is the software implementation of the system for increasing the reliability of data storage in the cloud.

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

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

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the Visual C# environment.

Keywords: computer science, reliability, cloud

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

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

ВСТУП

Актуальність теми. Ріст обсягів даних, що зросли вимоги до надійності зберігання й швидкодії доступу до даних роблять необхідним виділення засобів зберігання в окрему підсистему Обчислювального Комплексу (ОК).

Роль і важливість системи зберігання визначаються постійно зростаючою цінністю інформації в сучасному суспільстві, можливість доступу до даних і керування ними є необхідною умовою для виконання бізнес-процесів.

Безповоротна втрата даних піддає бізнес серйозній небезпеці. Втрачені обчислювальні ресурси можна відновити, а втрачені дані, при відсутності грамотно спроектованої й впровадженої системи резервування, уже не підлягають відновленню.

По даним Gartner, серед компаній, що постраждали від катастроф і які пережили велику необоротну втрату корпоративних даних, 43% не змогли продовжити свою діяльність.

Доступ до даних неможливий як у випадку виходу з ладу каналів (доступу) або обчислювальних засобів, так і у випадку відсутності необхідної продуктивності для виконання прикладних завдань.

Виділення засобів зберігання даних в окрему підсистему в рамках Обчислювального Комплексу дозволить проектувальникам сконцентруватися на рішенні проблем забезпечення надійного зберігання й доступу до даних у рамках однієї підсистеми. Крім того, це створює передумови для оформлення системи зберігання даних (СЗД) в організаційно-технічну структуру, що є основою для аутсорсингу послуг з надання засобів зберігання даних.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі.

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

– Огляд існуючих систем підвищення надійності зберігання даних у хмарі.

– Дослідження системи підвищення надійності зберігання даних у хмарі.

– Програмна реалізація системи підвищення надійності зберігання даних у хмарі.

Об'єктом дослідження є процес підвищення надійності зберігання даних у хмарі.

Предметом дослідження є методи підвищення надійності зберігання даних у хмарі.

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

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

– Удосконалено метод підвищення надійності зберігання даних у хмарі.

– Розроблено вітчизняний продукт підвищення надійності зберігання даних у хмарі, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі підвищення надійності зберігання даних у хмарі.

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

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти «Наука – виробництву», 2023, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №14.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Програмне забезпечення, яке розробляється у даному магістерському проекті, призначено для реалізації системи підвищення надійності зберігання даних у хмарі. Для багатьох система зберігання даних асоціюється із пристроями зберігання й, у першу чергу, з дисковими масивами. Дійсно, дискові масиви сьогодні є основними пристроями зберігання даних, однак, не варто забувати, що обробка інформації, формування логічної структури її зберігання (дискових томів і файлових систем) здійснюється на серверах. У процес доступу до даних, (крім процесорів і пам'яті сервера) залучені встановлені в ньому адаптери (Host Bus Adapter – HBA), що працюють за певним протоколом, драйвери, що забезпечують взаємодію HBA з операційною системою, менеджер дискових томів, файлова система й менеджер пам'яті операційної системи.

Якщо дисковий масив виконаний у вигляді окремого пристрою, то для його підключення до серверів використовується певна інфраструктура. Залежно від протоколу доступу (транспорту), реалізованого в HBA і дисковому масиві, вона може бути простою шиною (як у випадку із протоколом SCSI), так і мережею (як у випадку із протоколом Fibre Channel (FC)). Якщо це мережу, що одержала назву "мережа зберігання даних" (Storage Area Network – SAN), те, як і покладено мережі, у ній використовується активне встаткування – концентратори й комутатори, що працюють за протоколом FC, маршрутизатори протоколу FC в інші протоколи (звичайно в SCSI). Таким чином, крім пристроїв зберігання даних до складу СЗД необхідно ще додати інфраструктуру доступу, що зв'язує сервера із пристроями зберігання.

Відповідаючи на запитання, де правильно провести рису, що відокремлює систему зберігання від серверного комплексу, пропонується розглядати систему зберігання даних як "чорний ящик". Тоді, для підключення сервера до системи

зберігання, досить установити в сервер HBA з необхідним протоколом, підключити його до системи зберігання й сервер відразу "побачить" свої дані – тобто за принципом "plug and play". Це ідеальна ситуація, до якої IT-індустрія, можливо, прийде в майбутньому. Сьогодні границю, що відокремлює систему зберігання даних від серверів, треба проводити на самих серверах вище рівня менеджера дискових томів. А чому саме так, можна переконатися на наступному прикладі: у системах, де потрібен високий рівень готовності, дисковий масив може вважатися єдиною точкою відмови (Single Point Of Failure – SPOF). Для ліквідації SPOF звичайно встановлюється другий масив, при цьому дані дзеркалюються на обидва масиви. Сьогодні одним з найпоширеніших засобів дзеркалювання є менеджер дискових томів (наприклад, VERITAS Volume Manager). Таким чином, менеджер дискових томів залучений у процес забезпечення відказостійкості системи зберігання даних і стає її компонентом.

1.2 Область застосування

Областю застосування системи є сервери у мережі. Се́рвер як комп'ютер – це комп'ютер у локальній чи глобальній мережі, який надає користувачам свої обчислювальні і дискові ресурси, а так само доступ до встановлених сервісів; найчастіше працює цілодобово, чи у час роботи групи його користувачів. Се́рвер як програма – програма, що надає деякі послуги іншим програмам (клієнтам). Зв'язок між клієнтом і сервером зазвичай здійснюється за допомогою передачі повідомлень, часто через мережу, і використовує певний протокол для кодування запитів клієнта і відповідей сервера. Серверні програми можуть бути встановлені як на серверному, так і на персональному комп'ютері, щоразу вони забезпечують виконання певних служб (наприклад, сервер баз даних чи веб-сервер).

Комп'ютер або програма, що установлена на цьому комп'ютері, здатні автоматично розподіляти інформацію чи файли під керуванням мережної ОС або у відповідь на запити, прислані у режимі on-line користувачами, і таким чином надавати послуги іншим комп'ютерам мережі (клієнтам).

Загальне призначення сервера

У більшості загального користування сервер фізичного комп'ютера (система комп'ютерної техніки) призначений запустити одну або декілька послуг (як приймаюча сторона) для задоволення потреб користувачів інших комп'ютерів в мережі. В залежності від обчислювальних послуг, які вона пропонує, це може бути сервер баз даних, файловий сервер, поштовий сервер, сервер друку, веб-сервер, ігровий сервер, або якийсь інший сервер. У контексті архітектури клієнт-сервер, сервер являє собою комп'ютерну програму, яка обслуговує запити інших програм – «клієнтів». Таким чином, сервер виконує деякі обчислювальні завдання від імені "клієнтів". Сервери часто надають основні послуги через мережу, або в приватних користувачів – всередині великої організації або громадським користувачам – через Інтернет. Мережевий сервер являє собою комп'ютер, призначений для обробки запитів і передачі даних на інші (клієнт) комп'ютери по локальній мережі або через Інтернет. Мережеві сервери зазвичай конфігуруються з додатковою пам'яттю і ємністю для обробки навантаження з обслуговування клієнтів.

В залежності від функціонального призначення розрізняють:

– файлові сервери ;

– проксі-сервери;

– FTP-сервери;

– Web-сервери;

– DNS-сервери;

– SQL-сервери;

– термінальні сервери;

– Інтернет-сервери та інші.

Ролі сервера

Роль – це функція сервера (наприклад почтовий, контролер домена тощо). Один сервер може відігравати декілька ролей одночасно. При реєстрації адміністратора на сервері майстер "Manage Your Server" допомагає добавити нові ролі або змінити існуючі.

Файловий сервер

Додавання ролі файлового сервера оптимізує сервер для підтримки спільних папок і зберігання файлів. Після додавання ролі файл-сервера, ви зможете призначати користувачам дискові квоти, використовувати службу індексації для пошуку файлів і навіть робити пошук документів у різних форматах на різних мовах, використовуючи в меню Start інструмент Search або новий веб-інтерфейс пошуку. WS2K3 пропонує масу нових можливостей для поліпшення обслуговування файлів: Тіньове копіювання (Shadow copy) – Резервне побайтове копіювання ранніх версій документів, що дозволяє користувачам скасовувати зроблені зміни в документах, що зберігаються на сервері. Покращена розподілена файлова система DFS – дозволяє створювати єдине логічне іменоване простір для безлічі загальних папок, розташованих на різних серверах. Тепер користувачам не потрібно запам'ятовувати, на яких серверах розташовані часто використовувані ними спільні папки. DFS в WS2K3 також надає службу реплікації файлів з вибором топології, що було недоступне в Win2K. Крім того, сервери WS2K3 можуть обслуговувати кілька коренів DFS. Служба тіньового копіювання томів (Volume shadow copy service) – Створює копію оригінальних загальних даних на заданий момент часу. Програми резервного копіювання можуть використовувати цю копію, щоб зробити папку загального доступу статичної, поки змінюються поточні документи. Крім того, ви можете переміщати тіньові копії на інші сервера для резервного зберігання, тестування та аналізу даних.

Сервер друку

Сервери друку використовуються для надання та управління доступом до принтерів. Роль сервера друку дозволяє управляти принтерами через веб-оглядач, друкувати через URL принтера, використовуючи протокол IPP, а також підключати принтери, використовуючи Point and Print. Microsoft зробила ряд розширень служби друку в WS2K3: Підтримка кластерів друку – автоматична реплікація драйверів принтерів по всіх серверів в кластері. Розширення в Active Directory – адміністратори можуть виносити на загал принтери в AD, щоб користувачі могли шукати принтери в залежності від місця розташування,

кольору і швидкості. Поліпшення безпеки – групові політики, що дозволяють адміністратору запобігати доступу клієнтів до спулера, якщо сервер не обслуговує друк.

Сервер застосунків

Коли ви налаштовуєте сервер в якості сервера застосунків, ви встановлюєте Internet Information Services (IIS) 6.0 і цілий ряд компонентів, наприклад, COM+ і ASP.NET. Microsoft оптимізувала IIS 6.0 з точки зору стабільності, керованості, швидкої розробки додатків і безпеки. Роль сервера застосунків WS2K3 забезпечує підтримку нових веб-служб і платформи .NET, зокрема служби Universal Description, Discovery and Integration (UDDI), а також Simple Object Access Protocol (SOAP) і Web Services Description Language (WSDL). Сервери застосунків часто конфігурують включаючи наступне:

Злиття ресурсів

Управління розподіленими транзакціями:

– захист;

– відмовостійкість.

Поштовий сервер

Дозволяє обслуговувати базові поштові скриньки ваших користувачів і дозволяє приймати і відправляти пошту з сервера. Вхідна пошта може зберігатися на сервері, а потім забиратися користувачем по протоколу POP3. Для ролі поштового сервера ви повинні мати: Активне з'єднання з інтернет Зареєстроване доменне ім'я Запис MX у провайдера для вашого поштового домену

Термінальний сервер

Після інсталяції ролі термінального сервера, ви можете дозволити користувачам підключатися до сервера і запускати на ньому програми так, як ніби ці програми були інстальовані на робочій станції клієнта.

Remote Access/VPN Server

Сервери віддаленого доступу і VPN надають точку входу в вашу мережу для віддалених користувачів. Використовуючи роль Remote Access / VPN Server, ви можете реалізувати протоколи маршрутизації для середовищ LAN і WAN. Ця роль підтримує модемні з'єднання і VPN через інтернет.

Domain Controller

Контролер домену містить базу даних Active Directory. Контролери домену надають служби аутентифікації для користувачів і комп'ютерів, а також керують доступом до мережевих ресурсів. Роль контролера домену замінює інструмент DCPROMO, який був в Win2K. Ця роль дозволяє додати контролер домену до існуючого домену, створити новий домен, створити нове дерево.

DNS Server

Служба DNS дозволяє перетворювати доменні імена (FQDN) в адреси IP. Версія DNS в WS2K3 включає службу динамічного DNS (DDNS), яка дозволяє комп'ютерам самим реєструватися в базі даних DNS.

DHCP Server

Сервер DHCP дозволяє клієнтам отримувати свій IP за потребою. Сервер DHCP також надає додаткову інформацію для конфігурації мережі – адреса серверів DNS, WINS і т.п.

Streaming Media Server

Потоковий сервер надає служби Windows Media Services мережевим клієнтам. Windows Media Services використовуються для управління і доставки мультимедійного контенту – потокового відео та аудіо – через інтранет або інтернет.

WINS Server

WINS дозволяє клієнтам NetBIOS перетворювати імена комп'ютерів в адреси IP. На відміну від DNS, що вимагає доменні імена, WINS спроектована для внутрішньої інтрамережі для дозволу простих імен NetBIOS. Хоча можна мати мережу Windows без NetBIOS і WINS, багато утиліти все ще залежать від бази даних WINS. Багато типів записів, наявні в WINS, відсутні в DNS. Ці типи дозволяють легко знаходити в мережі сервери, які виконують специфічні служби (включаючи Terminal Services). Такий утилітою є Terminal Server Administration. Без WINS вам доведеться вручну вказувати сервер для управління.

Ігровий сервер – програмний компонент обчислювальної системи, що забезпечує зв'язок між різними клієнтами, надаючи їм

можливість комунікації один з одним в рамках програмної оболонки конкретної гри.

Серверне обладнання

Вимоги до обладнання для серверів варіюються залежно від сервера додатків. Абсолютна швидкість процесора не настільки важлива для сервера, як і для настільного комп'ютера. Обов'язки сервера надавати послуги багатьом користувачам по мережі призводять до різним вимогам, таких як швидке підключенням до мережі та висока пропускна спроможності. Так як сервери, як правило, доступні по мережі, вони можуть працювати без монітору. Процеси, які не потрібні для функції сервера не використовуються. Багато серверів не мають графічного інтерфейсу користувача. Крім того, аудіо-та USB інтерфейси можуть бути опущені. Сервери часто працюють протягом тривалого часу без перерви, тому надійність обладнання і довговічність надзвичайно важлива. Хоча сервери можуть бути побудовані з частин комп'ютера, критично важливі корпоративні сервери не можливі без використання спеціалізованого устаткування з низьким рівнем збою в цілях максимального часу безперебійної роботи, оскільки навіть короткострокові невдачі можуть коштувати дорожче, ніж покупка і установка системи . Наприклад, це може зайняти всього декілька хвилин часу простою на національній фондовій біржі, щоб виправдати рахунок повністю замінити системи з чимось більш надійним. Сервери можуть включати в себе більшу ємність жорстких дисків, більше комп'ютерних вентиляторів або водяного охолодження, щоб допомогти усунути тепло і джерела безперебійного живлення, які забезпечують роботу сервера в разі збою живлення. Ці компоненти забезпечують більш високу продуктивність і надійність за відповідно більш високою ціною.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Надійна файлова система

Перейдемо на один рівень вище, до файлової системи. Від її чекають простої речі – можливості записати файли й потім прочитати те, що записано. Парадоксально, але більша частина ФС цього не гарантує: вони покладаються на ідеальну роботу встаткування – дискового контролера, кабелю, самого диска. Апаратний збій приводить не просто до втрати даних – він приводить до непоміченого користувачем втраті даних. Копіюєте свій фотоархів – а насправді частина файлів уже знищена. Довідатися про це можна, звіривши контрольні суми, що ми робимо при перекачуванні прошивань і подібних не терплять збою даних. А чому б не звіряти контрольні суми засобами ФС?

Інше «дитяче» побажання до ФС – щоб вона працювала й не ламалася – теж толком не виконується. ФС без журналювання, наприклад FAT або ext2, при збої здатна поховати весь свій уміст. Журналюємі ФС, наприклад NTFS або ext3, істотно надійніше, тому що можна знайти крапку несуперечливого стану й відтворити журнал. А чи не можна створити ФС, що взагалі не може потрапити в суперечливий стан? Можна – через Copy-on-Write. Дані пишемо не поверх старих, а виділяємо новий блок, пишемо туди, і якщо все в порядку – заміняємо покажчик зі старих даних на нові.

ZFS

ZFS поєднує функціональність ФС і підтримку RAID-подібних масивів. Поширюється за вільною ліцензією (CDDL). Створена Sun Microsystems для Solaris. Портирована в FreeBSD починаючи з версії 7.0. Зовсім недавно проект

ZOL (ZFS on Linux) досяг стадії релізу. Можна чекати швидкого поширення ZOL, процес уже пішов. Існує й проект під Mac OS X, ZEVO, див. відповідний матеріал.

Придивитеся до ZFS, якщо вам цікава ФС із такими можливостями:

– зберігає контрольні суми й не дозволяє вважати сміття замість даних;

– цілісність, що зберігає, настільки, що утиліти начебто chkdsk або fsck для неї просто немає;

– постачена інструментом перевірки цілісності холодних даних і їхньої автоматичної корекції, якщо дані збережені з надмірністю;

– здатна миттєво створювати знімки свого стану й зберігати їх хоч за кожну хвилину місяця, монтувати будь-який набір знімків, відкочуватися до знімка.

Придивитеся до ZFS, якщо вам цікавий RAID:

– програмний, тобто не потребуючого апаратного контролера;

– апаратно-незалежний, були б SATA-порти;

– без дири по запису;

– здатний до реконструкції деградованого масиву з дисками, що не читаються частково, із втратою тільки тих даних, для яких немає жодної копії;

– працюючий при перевірці цілісності й реконструкції тільки з корисними даними, а не з усім масивом;

– с підтримкою аналогів RAID1 (дзеркало), RAID5 (надмірність у розмірі одного диска), RAID6 (двох) і навіть «RAID7» (який зберігає дані при виході з ладу будь-яких трьох дисків масиву), а також більше складних варіантів, подібних RAID50 або RAID60.

В ZFS є й недоліки, основні з яких:

– Наростити RAID-Z-Масив на один диск не можна. Можна замінити всі терабайтні диски на тритери – і збільшити обсяг. Можна зібрати з 3 (і більше) дисків ще один RAID-Z і додати його до існуючого, одержавши єдиний пул. Але

перетворити RAID-Z1 з 5 дисків в RAID-Z1 з 6 можна, тільки злив кудись інформацію, зруйнувавши масив і створивши новий.

– Масив не можна зменшити. Можна тільки збільшувати – додавати групи дисків, міняти диски на більші.

– Ресурсоємність. ZFS постійно вважає контрольні суми, що створює навантаження на процесор і використовує під кеші пам'ять. Удома в мене працювало з Atom 330 і 2 ГБ пам'яті. Хоча при використанні ZFS цього Атома для повної утилізації гигабітної мережі мені не вистачало, але 40-50 МБ/с багатьох улаштує.

– Так, ще: якщо зруйнувати ZFS і створити з тих же дисків новий масив (тобто дати пари команд і/або понатискати кнопки у веб-інтерфейсі, в обох випадках ігноруючи попередження), то дані зі зруйнованої ФС будуть надійно поховані – на відміну від, наприклад, NTFS, дані з якої відносно просто відновити й після перетворення таблиці розділів. Чи вважати це недоліком – залежить від точки зору.

ReFS + Storage Spaces

В Windows Server 2012 Microsoft запропонувала ФС ReFS і систему керування томами Storage Spaces. Зв'язування ReFS і Storage Spaces можна вважати певною мірою аналогом ZFS, причому із властивостями, не реалізованими в останній. Цікаві гнучкість видалення-додавання дисків, thin provisioning і ін. Недоліки теж є – це власницька ліцензія, доступна тільки в складі Windows Server 2012 і Windows 8.1.

Погано, що, судячи від відкликанням тих, хто тестував, продуктивність зв'язування ReFS – Storage Spaces відчутно, у рази, падає при використанні варіантів з парністю, тобто, грубо, аналога RAID5 або ZFS RAID-Z. Ще гірше, що з'явилися перші повідомлення про втрату даних через збої ReFS, причому офіційна підтримка не вирішила проблем. Так що цікавому рішенню, схоже, варто дозріти.

Btrfs + mdadm

Офіційно Btrfs для Linux поки не одержала статусу релізу, хоча розробляється з 2007 р. Подібно переважній більшості ФС (і на відміну від ZFS), це саме файлова система, створювана поверх блокового пристрою. Щоб одержати, грубо, аналог ZFS, потрібно додати, наприклад, добре налагоджений mdadm, одна з можливостей якого – перебудова RAID при додаванні диска без втрати даних.

Традиційні ФС

І, звичайно, існування просунутих ФС не змушує вас вибрати одну з них. Можливо, вам для NAS більше підійде NTFS, ext4 або UFS – рідна ФС для обраної вами осі. На відносно слабкому залізі цей вибір може виявитися єдиним.

Хочу порекомендувати скачати VirtualBox, установити туди кілька варіантів і протягом декількох днів попрацювати з ними. На виртуалці це дешевше, простіше й швидше, ніж на реальному залізі. Серйозно заощаджує нерви. Особливо якщо жоден з варіантів вам не сподобається.

Як софт самозбірного NAS можна використовувати три групи програмних продуктів:

– По-перше, можна поставити повну операційну систему. Це може бути Linux, UNIX або Windows залежно від особистих потреб і переваг. Windows представляється більше знайомій, *nix дає більше NAS за ті ж гроші, але конфігурування *nix з нуля – завдання не для новачка.

– Для такого новачка створений варіант номер два – преконфігуровані спеціально під NAS збірки. Вони включають набір сервісів і веб-інтерфейс, що дозволяє новачкові впоратися з налаштуванням і використанням продукту без командного рядка.

– Нарешті, третій варіант – установка повної операційної системи, а поверх її – якогось веб-інтерфейсу для налаштування й керування. Варіант компромісний, проміжний. Спрощує використання, але ніяк не до рівня, достатнього для недосвідченого аматора.

Windows

У Windows як вісь NAS є не тільки достоїнства, але й недоліки:

– Windows – платний софт, що конкурує з багато в чому більше функціональним для конкретних завдань вільним.

– Підтримує існування вірусів, що вимагає постійного завантаження антивірусних баз.

– Вимагає постійного застосування апдейтів, а після них часто потрібна перезавантаження. Не можна сказати, що всі * nix-системи можуть працювати без втручання й перезавантаження роками. Але багато хто – можуть.

– Windows – графічна система, для природного її функціонування потрібний монітор, клавіатура й миша. NAS звичайно їх позбавлений. Але звичайно, існують засобу, що вирішують проблему.

– Windows вимогливий до ресурсів.

– Дискові масиви й контрольні суми на рівні файлової системи – не найдужче місце Windows, особливо в настільних варіантах.

Недоліки не смертельні, їх можна перебороти, обійти, ігнорувати, зрештою. Але достатні, щоб розглянути й інші варіанти. І виявити в них свої достоїнства, і зробити масу відкриттів, іноді не бажаючи того.

Приводячи ж неповний список достоїнств Windows, крім звичності можна згадати:

– Рідна високошвидкісна реалізація SMB.

– Сумісність із залізом. Драйвера для Windows пишуть обов'язково.

– Широкий вибір софту. Тобто для NAS звичайно великого вибору й не потрібно. Але чим більше специфічні ваші вимоги, тим більше ймовірність, що їх удасться вирішити саме софтом під Windows.

– Величезна безліч інструкцій і рішень на всі випадки життя, рідною мовою.

– NAS на Windows цілком можливий. Але тема налаштування Windows розкрита в багатьох джерелах.

Готові * nix-збірки для NAS

Найбільш популярні в профільній вітці три збірки – NAS4Free, OMV (=openmediavault) і FreeNAS 8.x. Усі націлені на середнім просуненні аматора й дозволяють або з коробки, або шляхом завантаження плагинів задовольнити типові потреби.

З торговельною маркою FreeNAS зв'язана історія, що викликає плутанину. Продукт FreeNAS розвивалися довгі роки, а потім з ряду причин розділився. Широко відома назва, що є торговельною маркою, потрапила у власність компанії iXsystems, що вирішила повністю переписати код, а вихідну розробку закрити. Так з'явився FreeNAS 8.x, заснований на FreeBSD 8.x. Однак оригінальний проект силами ентузіастів вижив, був перенесений на FreeBSD 9.x і успішно розвивається під ім'ям NAS4Free. Як результат, оновлення зі збереженням налаштувань із FreeNAS 0.7 підтримується в NAS4Free, але не підтримується в FreeNAS 8.

І, природно, потрібні диски для зберігання даних. Схоже, розроблювачі FreeNAS указують не вимоги, щоб тільки початок працювати, а щоб працювало продуктивно. У всякому разі, у посібнику з експлуатації при установці на віртуальну машину зазначений мінімальний розмір оперативної пам'яті в 512 МБ.

NAS4Free

NAS4Free поширюється за вільною ліцензією BSD. Проект має багаторічну історію (походить від m0n0wall, FreeNAS 0.7, 0.6 і раніше), старша цифра поточної версії – 9. Тобто збірки непогано вилизане, але продовжує розвиватися. Заснована на останньому релізі FreeBSD 9.1.

NAS4Free скомпільована у двох версіях, для 32– і 64-розрядних Intel x 86-сумісних процесорів. Використовувати NAS4Free можна трьома способами: LiveCD/LiveUSB, full і embedded. Режим «живого диска» традиційний для * nix-дистрибутивів і призначений насамперед для ознайомлення. При реальному використанні звичайно роблять установку. Варіант full – традиційна установка, звичайно на жорсткий диск. Для установки використовується невеликий розділ,

розмір якого задається при установці, і swap-розділ, а інша частина диска доступна для даних. Поставити full на USB-флешку можна, але через інтенсивний запис флешка за кілька місяців зноситься.

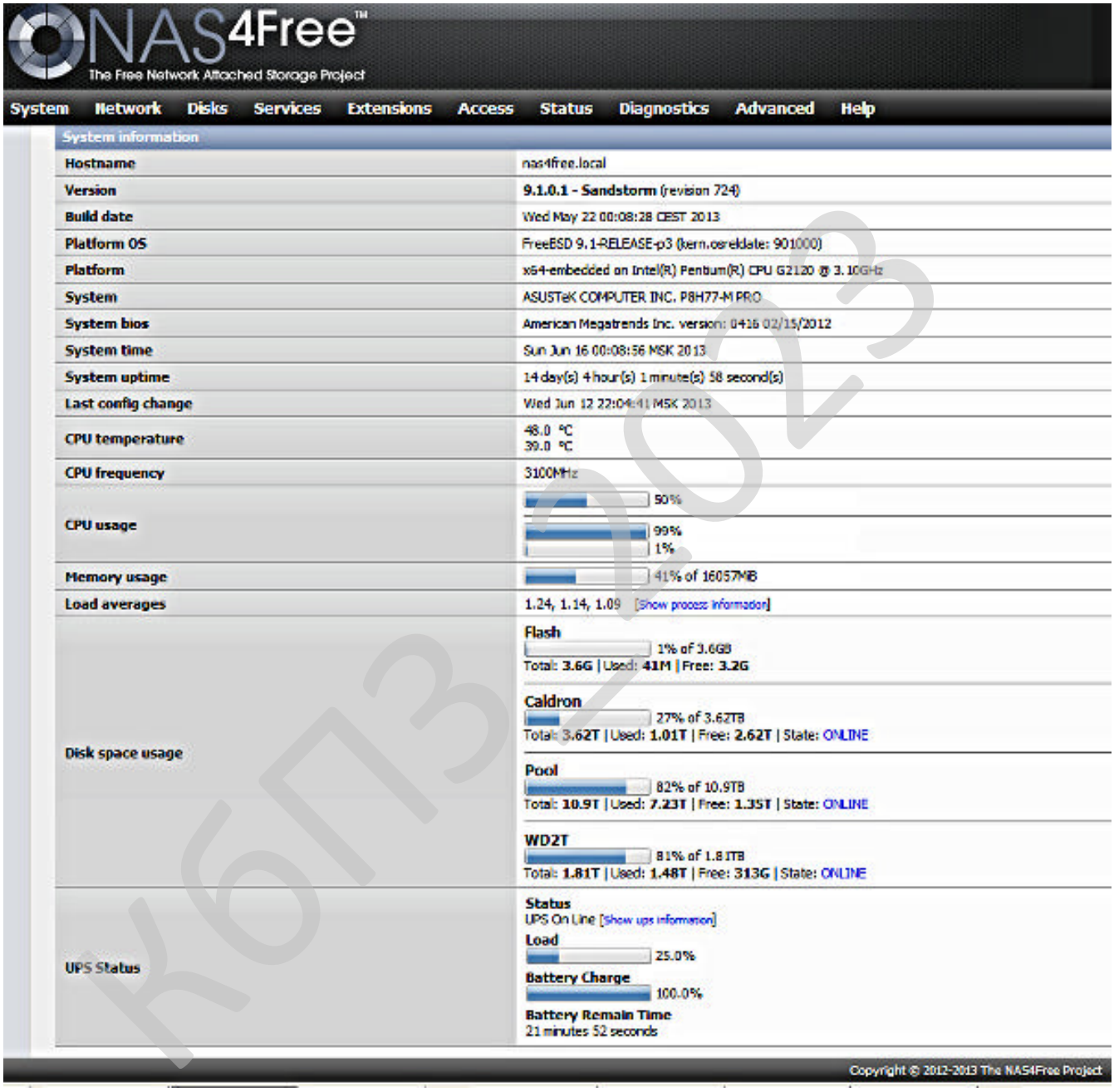

Рисунок 2.1 – Вид веб-інтерфейсу NAS4Free

Для установки на USB-флешку призначений варіант установки embedded. При старті системи створюється невеликий диск у пам'яті, куди копіюється образ

системи й відразу настроюється по параметрах, зібраним у єдиний конфігураційний файл XML. А потім виробляється завантаження із цього диска в пам'яті. Такий підхід має достоїнства. Систему дуже зручно розвертати – системну флешку можна записати на іншій машині. Стан системи зібраний в одному текстовому файлі, так що його дуже легко зберігати й, при необхідності, використовувати для відновлення системи. Диск у пам'яті дуже швидкий, а системна флешка практично не зношується.

Недолік у тому, що майже будь-які зміни, внесені в систему інакше, чим через веб-інтерфейс, губляться після перезавантаження. Частково розроблювачі вирішили проблему, надавши у веб-інтерфейсі можливість збереження безлічі параметрів. Частково проблему можна обійти за рахунок застосування трюків начебто об'єднання через unionfs папки диска в пам'яті й папки на реальному носії й використання запускаємих автоматично командних скриптів. Більше просунуті користувачі використовують віртуалізацію, щоб ставити в jail або окремі віртуальні машини всі що завгодно, хоч пари-трійку Windows. Розширення TheBrig серйозно полегшує налаштування й використання jail для установки додаткового софту. Але це все-таки вимагає знань, порівнянних з необхідними для розгортання повної системи. А новачкові вірніше вважати, що в NAS4Free є тільки ті сервіси, що включено споконвічно, усе настроюється через веб-інтерфейс, і додавати нічого не можна.

FreeNAS

FreeNAS – збірка на базі FreeBSD, створена й розвивається компанією iXsystems. Компанія розробляє також платну версію. Довгий час безкоштовна FreeNAS була штучно серйозно обмежена. Версія FreeNAS, істотно дороблена й забезпечує необхідну для домашнього NAS функціональність. В FreeNAS прекрасна англомовна документація. Комерційних корінь продукту простежуються й у високих апаратних вимогах (6+ ГБ RAM), і фокусі на використовувану в бізнесі функціональність. Типово домашні сервіси, такі як біторрент і UPn/ DLNA-Медіасервер, реалізуються як додаткові модулі, кожний

у власній клітці (jail). Робота з jail ведеться через користувальницький інтерфейс і, як і весь проект, прекрасно документований.

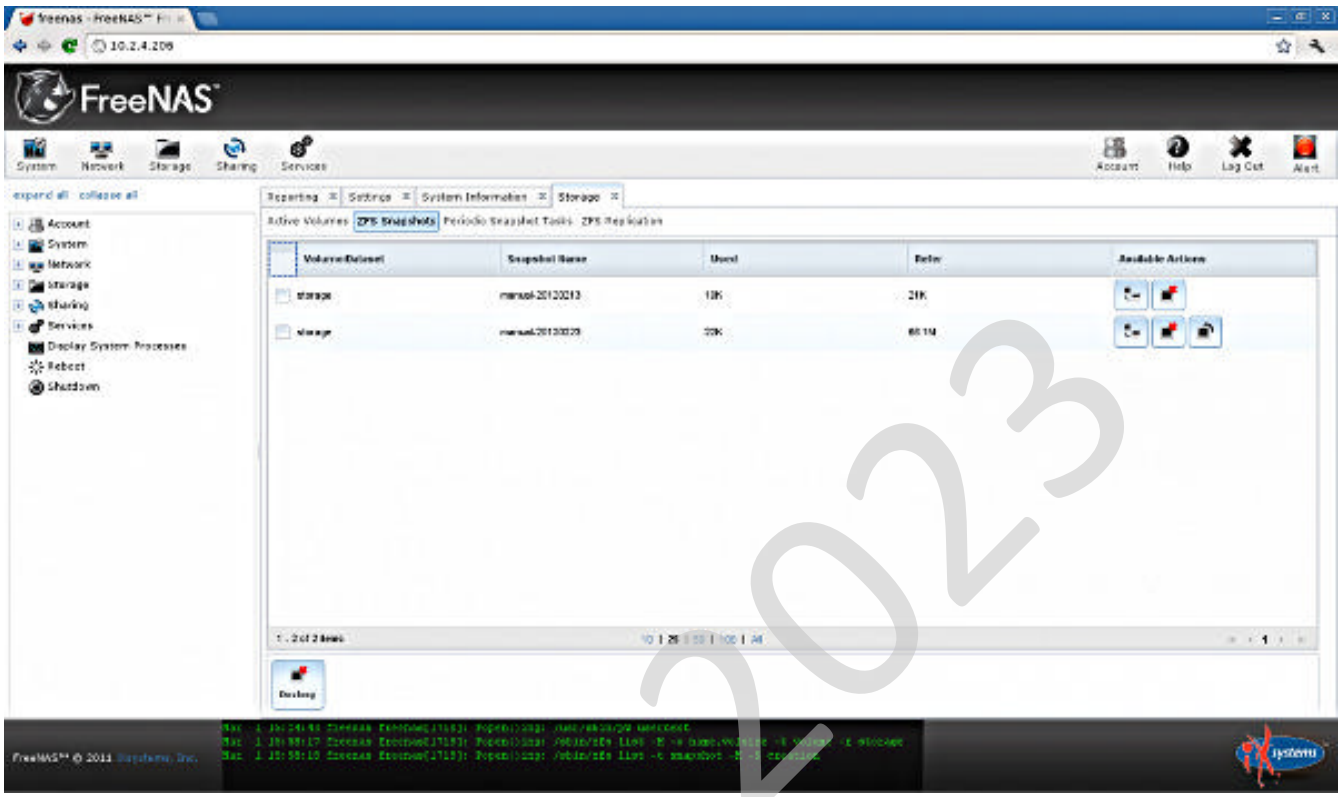

Рисунок 2.2 – Вид веб-інтерфейсу FreeNAS 8

FreeNAS установлюється на флешку обсягом не менш 2 ГБ і займає її цілком. Використовується nanoBSD-образ, що розвертається в оперативну пам'ять, що охороняє флешку від зношування. Хоча існує й 32-розрядна версія, вона скоріше призначена для ознайомлення. Для реальної експлуатації рекомендується 64-розрядна версія, зокрема через вимоги до обсягу пам'яті, несумісних з 32-розрядною архітектурою.

Функціональність «з коробки» близька до функціональності NAS4Free. Відмінність у доступності двох UPn-серверів на вибір (FUPPES і miniDLNA) і відсутності веб-сервера (розроблювачі обіцяють згодом додати його).

Створені в NAS4Free дискові ZFS-пули можуть бути імпортовані в FreeNAS і навпаки зі збереженням даних. У профільній вітці користувачів

FreeNAS небагато, і деякі скаржаться на проблеми (див. FAQ вітки). Втім, ще не створено програмного продукту, на який би хоч хтось не скаржився.

OMV

OMV заснований на Debian Linux, що забезпечує найбагатшу функціональність і розширюваність.

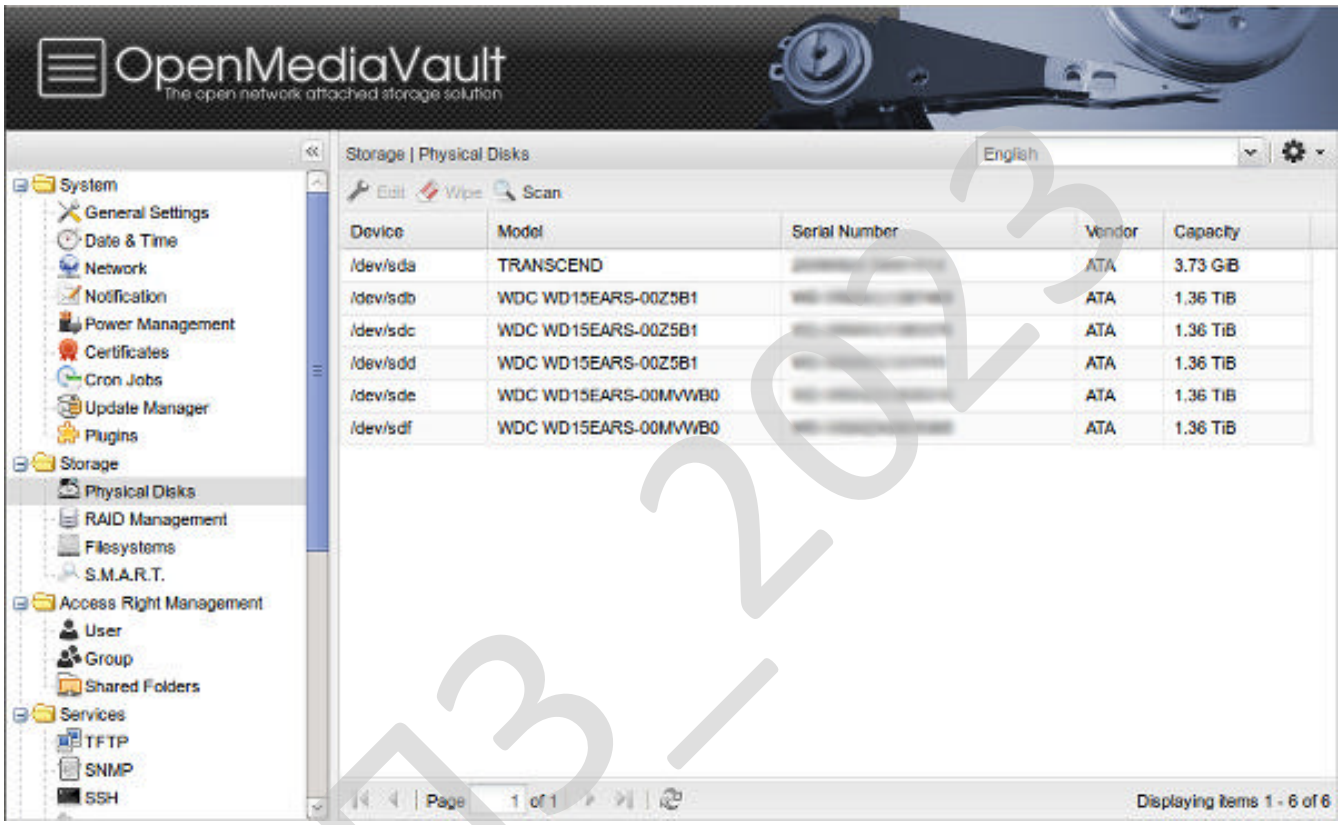

Рисунок 2.3 – Вид веб-інтерфейсу OMV

Продукт створений і підтримується Volker Theile, у минулому одним із провідних розроблювачів FreeNAS. Поширюється за відкритою ліцензією GPL. Підтримки ZFS у продукті немає, і вона, за заявою розроблювача, не планується, що й представляється його головним недоліком. У мережі можна знайти хак по установці zfsonlinux на OMV. OMV установлюється на жорсткий диск або USB-флешку, при цьому займає носій повністю. Така інсталяція не виглядає оптимальною ні для диска, ні для флешки. Дивно використовувати весь жорсткий

диск під систему, який потрібний усього гігабайт. Боязно встановлювати на флешку продукт, інтенсивно на неї пишучий. Однак для обох проблем ентузіастами розроблені хаки, що вирішують ці проблеми: спеціальний скрипт виносить каталоги з інтенсивним записом до пам'яті, а нескладне редагування дистрибутива дозволяє створити на системному диску розділ для даних. Описувати функціональність OMV особливого змісту немає. Базова функціональність, аналогічна NAS4Free і FreeNAS, за примітною відсутністю ZFS, доступна з коробки або як набір додаткових модулів. Але OMV – по суті своєї Debian, варіант Linux, для якого існує неймовірна безліч додатків. Їх можна встановлювати стандартним для Linux способом.

Важливою перевагою OMV над згаданими вище складаннями є можливість утилізації гігабитного каналу на досить слабкому залізі, такому як Intel Atom з гігабайтом пам'яті.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

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

Синтаксис Visual C# дуже виразний, але простий у вивченні. Усі, хто знаком з мовами C, C++ або Java з легкістю взнають синтаксис із фігурними

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

Як об'єктно-орієнтована мова, Visual C# підтримує поняття інкапсуляції, спадкування й поліморфізму. Всі змінні й методи, включаючи метод `Main` – крапку входу застосунка – інкапсулюється у визначення класів. Клас може успадковувати безпосередньо з одного родового класу, але може реалізовувати будь-яке число інтерфейсів. Для методів, які перевизначають віртуальні методи в батьківському класі, необхідно ключове слово override, щоб виключити випадкове повторне визначення. У мові Visual C# структура схожа на полегшений клас: це тип, що розподіляється по стопках, що реалізує інтерфейси, але не підтримуюче спадкування.

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

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

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

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

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

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

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

Архітектура платформи .NET Framework

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

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

При виконанні програми на Visual C# складання завантажується в середовище CLR залежно від відомостей у маніфесті. Далі, якщо вимоги безпеки

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

Взаємодія між мовами є ключовою особливістю .NET Framework. Оскільки код IL, створюваний компілятором Visual C# відповідає специфікації CTS, код IL на основі Visual C# може взаємодіяти з кодом, створюваним версіями мов Visual Basic, Visual C++, Visual J# платформи .NET Framework і ще більш ніж 20 CTS-сумісних мов. В одному складанні може бути кілька модулів, написаних на різних мовах платформи .NET Framework, і типи можуть посилатися один на одного, як якби вони були написані на одній мові.

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

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи підвищення надійності зберігання даних у хмарі.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Мережною інфраструктурою, що поєднує велику кількість серверів і пристроїв зберігання, необхідно управляти й, як мінімум, відслідковувати її стан. Сказане не означає, що немає необхідності моніторингу стану, наприклад, двох серверів і одного масиву, підключеного до них прямо. Однак, це можна реалізувати підручними засобами – убудованими утилітами серверів, масиву й операційної системи, безкоштовними (freeware) утилітами або скриптами. Кожне із пристроїв у СЗД має кілька об'єктів, що вимагають керування й контролю стану, наприклад дискові групи й томи в масивів, порти в масивів і комутаторів, адаптери в серверах. Як тільки число об'єктів керування в СЗД починає обчислюватися десятками, керування такою конфігурацією за допомогою "підручних" засобів віднімає в адміністраторів занадто багато часу й сил, і неминуче приводить до помилок. Упоратися з таким завданням можна тільки використовуючи повномасштабну систему керування. Це справедливо для будь-яких великих систем і для великої системи зберігання даних, зокрема. Впровадження системи керування стає особливо актуальним у тих випадках, коли система зберігання даних виділена не тільки структурно й функціонально, але й організаційно.

Система зберігання даних повинна включати наступні підсистеми й компоненти:

– Пристрої зберігання даних: дискові масиви й стрічкові бібліотеки. Сучасні високопродуктивні дискові масиви використовують технологію Fibre Channel для підключення до них серверів і для доступу до дисків усередині масиву. Вони можуть масштабуватися до десятків терабайт дискового простору й мають убудований інтелект для виконання спеціальних

функцій, таких як: віртуалізація дискового простору, розмежування доступу до дискового простору, створення Point-In-Time (PIT) копій даних і реплікація даних між масивами. До пристроїв зберігання даних також ставляться всілякі бібліотеки – стрічкові, магнітооптичні й CD/DVD, які в розглядатися не будуть. Визначення поняття Point-In-Time копії даних (PIT-копія, іноді зустрічається скорочення I-T-копія) треба з його назви – це копія даних, зроблена на певний момент часу, і стан даних "заморожено" у момент створення копії. Іноді плутають PIT-копії з "моментальними знімками" (SnapShot), які в дійсності є тільки одним з методів створення PIT-копій. До інших методів створення PIT-копій ставляться методи клонування (clone) даних.

– Інфраструктуру доступу серверів до пристроїв зберігання даних. У цей час, як правило, інфраструктура доступу серверів до пристроїв зберігання даних створюється на основі технології SAN. SAN є високопродуктивною інформаційною мережею, орієнтованої на швидку передачу більших обсягів даних.

– В основі концепції SAN лежить можливість з'єднання кожного із серверів з будь-яким пристроєм зберігання даних, що працюють за протоколом Fibre Channel. Мережа зберігання даних утворять: волоконно-оптичні з'єднання, Fibre Channel Host Bus Adapters (FC-HBA) і FC-комутатори, у цей час передачі, що забезпечують швидкість, 200 Мбайт/с і далекість між об'єктами, що з'єднуються, до декількох десятків кілометрів. У випадку, якщо відстань між об'єктами перевищує можливості FC-устаткування ні достатньої кількості "темної" оптики, зв'язок між об'єктами можна забезпечити використовуючи технологію ущільненого спектрального мультиплексування DWDM або інкапсулював FibreChannel в інший транспортний протокол, наприклад в TCP/IP. Технологія DWDM (Dense Wavelength Division Multiplexing) дозволяє оптимальним образом застосовувати оптоволоконі ресурси й передавати не тільки трафік Fibre Channel, але також Ethernet і інші протоколи по тим самим оптичних каналах одночасно. При цьому відстані між об’єктами, що з’єднуються,

можуть становити сотні й навіть тисячу кілометрів. Докладніше про SAN можна прочитати в [1]. "Темна" оптика – це технічний жаргон, що позначає оптичну магістраль (кабель) на шляху проходження якої не встановлені ніякі активні пристрої. Відсутність таких пристроїв має на увазі, що по кабелю не передається ніяких сигналів. Для оптики таким сигналом є світло, тобто в оптичний кабель не світить ні який пристрій. Звідси й походження терміна. Без застосування додаткових пристроїв, наприклад FC-ATM конвертерів, FC-комутатори не можуть віддавати пакети по магістралі, де присутні інші активні пристрої.

– Систему резервного копіювання й архівування даних. Система призначена для створення резервних копій і відновлення даних. Система резервного копіювання дозволяє захистити дані від руйнування не тільки у випадку збоїв або виходу з ладу апаратури, але й у результаті помилок програмних засобів і користувачів.

– Виконання резервного копіювання є одним з необхідних методів забезпечення безперервності бізнесу. Створення централізованої системи резервного копіювання дозволяє скоротити сукупну вартість володіння IT-інфраструктурою за рахунок оптимального використання пристроїв резервного копіювання й скорочення витрат на адміністрування (у порівнянні з децентралізованою системою).

– Програмне забезпечення керування зберіганням даних. Програмне забезпечення призначене для рішення завдань керування зберіганням даних, наприклад, для розмітки дискових томів або підвищення продуктивності доступу до даних прикладного ПЗ. Наприклад, убудоване в масиви Hitachi Lightning 9900V програмне забезпечення Cruise Control збирає статистику по інтенсивності роботи з даними, і виходячи з її ухвалює рішення щодо переміщенні даних на диски, продуктивність яких відповідає швидкості звертання до даних.

– Систему керування. Система призначена для моніторингу й керування рівнем якості сервісу зберігання даних. Вона тісно інтегрується із системою керування ОК. Основою системи керування СЗД є засоби керування апаратними

ресурсами мережі зберігання даних. Їхня інтеграція з іншими системами дає можливість контролювати ресурси СЗД і управляти ними на всіх рівнях, від дисків у масиві до файлової системи сервера.

Серед підсистем СЗД система резервного копіювання заслуговує на особливу увагу. Як треба з визначення, створення системи резервного копіювання є одним із засобів забезпечення надійного зберігання даних, про які поговоримо нижче. Однак, систему резервного копіювання необхідно включити в СЗД як окрему підсистему не тільки із цієї причини. Обсяг даних, вимірюваний одиницями й десятками терабайтів, вимагає усе більше часу на процедуру резервного копіювання. Класичні засоби резервного копіювання по ЛОМ не встигають виконати цю процедуру й укластися у відведене тимчасове "вікно", що скорочується з наближенням режиму роботи інформаційної системи до "24x7" (наприклад, у системах обслуговуючі регіони із центра). Рішенням зазначеної проблеми є використання SAN для передачі даних резервного копіювання, а також застосування засобів сучасних дискових масивів для створення PIT-копій. У цьому випадку буде потрібно тісна інтеграція системи резервного копіювання з SAN і дисковими масивами.

Завдання які стоять перед системою зберігання даних

Система зберігання даних призначена для організації надійного зберігання даних, а також відказостійкого, високопродуктивного доступу серверів до пристроїв зберігання. Існуючі в цей час методи по забезпеченню надійного зберігання даних і відказостійкого доступу до них можна охарактеризувати одним словом – дублювання.

Так, для захисту від відмов окремих дисків використовуються технології RAID, які (крім RAID-0) застосовують дублювання даних, збережених на дисках. Рівень RAID-5 хоча й не створює копій блоків даних, але все-таки зберігає надлишкову інформацію, що теж можна вважати дублюванням. Для захисту від логічного руйнування даних (руйнування цілісності бази даних або файлової системи), викликаних збоями в устаткуванні, помилками в програмному

забезпеченні або невірних діях обслуговуючого персоналу, застосовується резервне копіювання, що теж є дублюванням даних. Для захисту від втрати даних внаслідок виходу з ладу пристроїв зберігання через техногенну або природну катастрофу, дані дублюються в резервний центр.

Відказостійкість доступу серверів до даних досягається дублюванням шляхів доступу. Стосовно до SAN дублювання полягає в наступному: мережа SAN будується як дві фізично незалежні мережі, ідентичні по функціональності й конфігурації. У кожний із серверів, включених в SAN, установлюється як мінімум по двох FC-HBA. Перший з FC-HBA підключається до однієї "половинці" SAN, а другий – до іншої. Відмова встаткування, зміна конфігурації або регламентні роботи на одній із частин SAN не впливають на роботу іншої. У дисковому масиві відказостійкість доступу до даних забезпечується дублюванням RAID-контролерів, блоків живлення, інтерфейсів до дисків і до серверів. Для захисту від втрати даних дзеркалюють ділянки кеш-пам'яті, що беруть участь в операції записи, а електроживлення кеш-пам'яті резервують батареями. Шляхи доступу серверів до дискового масиву теж дублюються. Зовнішні інтерфейси дискового масиву, включеного в SAN, підключаються до обох її "половинок". Для перемикання зі шляху, що вийшов з ладу, доступу на резервний , а також для рівномірного розподілу навантаження між всіма шляхами, на серверах установлюється спеціальне програмне забезпечення, що поставляється або виробником масиву (EMC CLARiiON – PowerPath, HP EVA – AutoPath, HDS – HDLM), або третім виробником (VERITAS Volume Manager).

Необхідну продуктивність доступу серверів до даних можна забезпечити створенням виділеної високошвидкісної транспортної інфраструктури між серверами й пристроями зберігання даних (дисковим масивом і стрічковими бібліотеками). Для створення такої інфраструктури в цей час найкращим рішенням є SAN. Використання сучасних дискових масивів з достатнім обсягом кеш-пам'яті й продуктивної, що не має "вузьких місць" внутрішньою архітектурою обміну інформацією між контролерами й дисками, дозволяє

здійснювати швидкий доступ до даних. Оптимальне розміщення даних (disk layout) по дисках різної ємності й продуктивності, з потрібним рівнем RAID залежно від класів додатків (СУБД, файлові сервіси й т.д.), є ще одним способом збільшення швидкості доступу до даних.

Disk layout – це схема розподілу дані додатки по дисках. Вона враховує в які рівні RAID організовані диски, число й розміри розділів на дисках, які файлові системи використовуються й для зберігання яких типів даних вони призначені.

Необхідно помітити, що оптимізація налаштувань програмних засобів, як самих додатків, так і операційної системи, дає істотно більший приріст продуктивності системи, ніж використання могутнішої апаратури. Обумовлено це в першу чергу тим, що оптимізація налаштувань усуває "вузькі місця" (bottleneck) на шляхах проходження потоків даних, тоді як нова апаратура робить "горлечко пляшки" ледве ширше й тільки (хоча іноді й цього досить для рішення проблем швидкодії). У рішенні завдання оптимізації може допомогти застосування спеціального ПЗ, у якому реалізовані функції, що враховують особливості взаємодії апаратури, операційної системи й прикладного ПЗ. Прикладом такого ПЗ служить опція Quick I/O файлової системи VxFS. Опція Quick I/O ліцензується в складі пакета VERITAS DataBase Edition (DBE) for ORACLE. Зазначена опція дозволяє СУБД ORACLE використовувати Kernel Asynchronous IO (KAIO) для доступу до файлів даних, що істотно підвищує продуктивність операцій уведення-виводу СУБД. Докладніше про це можна прочитати в [4].

Крім досягнення необхідних показників продуктивності, відказостійкості й надійності зберігання даних у СЗД, замовники також прагнуть скоротити сукупну вартість володіння системою (Total cost of ownership – TCO). Впровадження системи керування дозволяє скоротити витрати на адміністрування СЗД і спланувати витрати на її модернізацію. Консолідація технічних засобів також сприяє скороченню витрат на експлуатацію СЗД.

Завдання які треба вирішити проектувальникові в процесі створення системи зберігання даних

У процесі створення СЗД необхідно досягти оптимального співвідношення продуктивності, доступності (надійного зберігання й відказостійкого доступу) і сукупної вартості володіння.

Одним з найбільше часто використовуваних методів забезпечення високої доступності СЗД є дублювання, що підвищує вартість системи. Якщо не враховувати бізнес-вимог замовника до доступності системи, то система стає невиправдано дорогою. Погоня за непотрібною продуктивністю також приведе до використання дорогих технічних засобів. Крім високих показників продуктивності, доступності й низкою вартості потрібно також забезпечити необхідну функціональність – обсяг зберігання й число серверів, що підключаються.

На жаль, замовники не завжди можуть описати вимоги по продуктивності в кількісних характеристиках, прийнятих для систем зберігання – пропускної здатності (Мбайт/с) або продуктивності (операцій уведення-виводу в секунду – I/O per second (IOPS)). Тому, щоб визначити якщо не кількісні характеристики, те хоча б характер навантаження, проектувальникові необхідно з'ясувати, роботу яких додатків повинна забезпечувати СЗД.

Різні класи додатків створюють різне навантаження на дискову підсистему. Наприклад, для СУБД характерні види навантажень, що залежать від класів завдань – транзакційні системи (Online Transaction Processing (OLTP)) і аналітичні системи (Decision Support Systems (DSS)). Для завдань класу OLTP характерним є великий потік запитів на уведення-вивід невеликих порцій даних. Для завдань класу DSS, навпроти, характерно невелике число запитів на читання більших порцій інформації.

Від того, яку навантаження дає додаток, залежить вибір способу розподілу даних по дисках і визначення обсягу кеш-пам'яті дискового масиву. Так для OLTP-завдань наявність кеш-пам'яті в дисковому масиві може зіграти істотну

роль для підвищення продуктивності вводу-виводу. Навпроти, у завданнях класу DSS відбувається зчитування більших обсягів даних, що практично виключає їхнє повторне влучення в кеш-пам'ять (на відміну від OLTP-завдань). Таким чином, кешування зчитувальних даних при обробці DSS-завдань не завжди збільшує їхню продуктивність.

До типів навантаження на СЗД, виробленими завданнями класу OLTP, DSS і файловими сервісами можна віднести інші відомі типи додатків. Так, поштовий сервіс, побудований на базі MS Exchange або Lotus Domino, дає подібне навантаження на СЗД, що й OLTP, оскільки зазначені продукти побудовані на базі СУБД. Поштовий же сервіс, заснований на sendmail, робить навантаження на СЗД, характерну для файлової системи із частою зміною метаданих. Засоби резервного копіювання виконують послідовне читання даному, підлягаючому резервному копіюванню, що робить характер їхніх операцій схожим з DSS-завданнями.

Існує ще один, не згаданий раніше, тип навантаження, характерний для процесів журналювання. Прикладом можуть служити запису журналів транзакцій СУБД або журналів подій операційних систем. До цього класу також можна віднести завдання завантаження даних у БД або сховище даних (Data Warehouse). Навантаження, здійснюване на СЗД цим класом завдань, аналогічна навантаженню DSS тільки для операцій запису. Тут наявність кеш-пам'яті (на запис) у дисковому масиві не збільшує продуктивності запису даних. Дану тезу необхідно пояснити. Звичайне застосування кеш-пам'яті на запис зменшує час, що сервер витрачає на операцію запису й очікування її завершення. Але при записі великого обсягу інформації (завантаження даних у БД) або при записі даних, які пишуться на нове місце, таких як журнал транзакцій, існує висока ймовірність виникнення ситуації, коли кеш-пам'ять на запис уже буде повністю зайнята, а новий записуваний блок не відповідає жодному із уже присутніх у кеш-пам'яті. У цьому випадку масиву прийде звільнити кілька блоків кеш-пам'яті, записавши їхній уміст на диски, перш ніж почати обробку операції, що надійшла.

Визначити класи завдань, які буде обслуговувати СЗД, необхідно не тільки для виробітку політики роботи з кеш, але також для розподілу даних по дисках (disk layout). Для забезпечення надійного зберігання інформації диски організуються в рівні RAID 1, 0+1/1+0 або 5. Самим економічним з погляду використання додаткового (дублюючого) дискового простору є рівень RAID 5. Однак продуктивність RAID 5 нижче, ніж в RAID 1 або 0+1 при частих випадкових змінах даних, характерних для OLTP-завдань, і висока при зчитуванні даних, що характерно для DSS-завдань.

Також різні рівні RAID забезпечують різні рівні відказостійкості дискової пам'яті до збоїв окремих дисків. Так RAID 5 не рятує від двох послідовних відмов дисків, втім, як і RAID 0+1, якщо це диски різних половин "дзеркала". Найбільше відказостійким є рівень RAID 1+0 (попарне "дзеркалювання" дисків і потім їх "striping"), тому його рекомендується застосовувати для зберігання критичних даних, наприклад, журналів транзакцій СУБД. Практика показує, що для зберігання файлових систем і даних DSS-завдань досить використовувати RAID 5. Однак, якщо файлова система часто змінюється як, наприклад, у поштових системах sendmail, те має сенс використовувати журнальовану файлову систему або файлову систему з окремо збереженими метаданими, наприклад файлову систему Sun QFS. Тоді для зберігання журналів або окремих метаданих краще застосовувати RAID 1 або 1+0.

Striping – метод розміщення даних на дисках, при якому послідовно, що йдуть блоки, даних, складові логічний том, записуються по черзі на кожний фізичний диск, що входить у дискову групу. У такий спосіб досягається більша продуктивність, оскільки операції читання й запису на диски можуть вироблятися паралельно в порівнянні з варіантом, коли всі блоки логічного тому записується на один диск. Докладніше про striping і його вплив на продуктивність уведення-виводу можна прочитати в [3].

Докладніше про вплив кеш-пам'яті й disk layout на продуктивність уведення-виводу завдань класу OLTP і DSS можна прочитати в [3].

Для "великих" систем актуальної стає оптимізація налаштувань СЗД, спрямована на підвищення швидкодії для досягнення заданого рівня сервісу. Під "великий" розуміється така система, у якій обробляється значний обсяг даних – одиниці й десятки терабайт, і/або до СЗД підключені десятки серверів. Для невеликих систем проблеми зі швидкодією можна вирішити застосуванням більше продуктивної апаратури. В "великих" системах такий підхід може виявитися неприйнятним або у зв'язку з тим, що буде потрібно дуже дорога апаратура, або у зв'язку з тим, що вже досягнуть межа апаратної продуктивності. Єдиним рішенням у цьому випадку є оптимізація. Для оптимізації продуктивності СЗД бажано мати можливість управляти налаштуваннями на всьому шляху проходження даних від процесора до дисків і назад. Для СУБД ORACLE така оптимізація полягає в можливості використовувати KAIO, а також можливості відключити кеш файлової системи для файлів даних ORACLE (оскільки СУБД ORACLE кешує дані у власній області пам'яті SGA). Для цієї мети можна використовувати згаданий раніше пакет VERITAS DBE. Якщо в системі експлуатуються OLTP– і DSS-завдання (що є типовим для більшості систем), то для оптимізації продуктивності дискового масиву бажано мати можливість управляти налаштуваннями кеш-пам'яті для кожного логічного диска (LUN) окремо. Це необхідно для того, щоб для тих дисків, де розташовані дані OLTP-завдання, використовувати кеш (і бажано великого обсягу), а для дисків з даними DSS-завдання використання кеш-пам'яті відключити. Однак, якщо для OLTP– і DSS-завдань використовуються ті самі таблиці даних, те такі налаштування втрачають зміст доти, поки не буде вирішене питання про фізичне рознесення даних завдань по різних дисках, а виконання самих завдань перенесено на різні сервери. Це можна реалізувати засобами СУБД, наприклад, за допомогою експорту даних у файл і імпорту їх в іншу базу. Якщо обсяг даних великий і синхронізацію баз даних OLTP– і DSS-завдань треба проводити досить часто, цей варіант може виявитися неефективним. Тут може допомогти технологія створення PIT-копій даних, реалізована в більшості сучасних дискових масивів.

Вище говорилося, що СЗД є важливою ланкою в забезпеченні заданого рівня сервісів, надаваних інформаційною системою. Рівень сервісу залежить не тільки від продуктивності СЗД, питання забезпечення якої тільки що обговорювалися, але також від готовності й надійного зберігання даних, інакше кажучи, від доступності СЗД. Виходячи з бізнес-вимог до інформаційної системи, необхідно визначити режими її роботи (24x7, 12x5 і т.п.), ступінь критичності даних залежно від ступеня критичності сервісів, що використовують ці дані, вимоги до готовності й надійності даних залежно від ступеня їхньої критичності й режимів роботи системи.

Розглянемо для приклада роботу інформаційної системи комерційного банку. У банку експлуатується Автоматизована Банківська Система (АБС), що обслуговує фінансові транзакції клієнтів банку (OLTP-завдання). Режим роботи банку 8: 00-20:00. Банк має кілька філій у ряді регіонів України, які працюють із АБС головного офісу в термінальному режимі. Робочі годинники АБС становлять 6: 00-22:00. Припустимий час простою АБС – не більше 1 години. Припустимо втратити дані АБС не більш ніж за 0,5 години, оскільки за цей період вони можуть бути повторно уведені в систему з паперових носіїв (фактично це обумовлено бізнес-вимогою за часом проходження фінансової транзакції). Також у банку експлуатуються аналітичні завдання (DSS) на основі даних з АБС. Робочий час аналітиків 9:00-18:00. Припустимий час простою сервісів аналітичних завдань не більше 4-х годин. Завантаження даних з АБС в аналітичну базу (синхронізація) відбувається по закритті "операційного дня" в 23:00. Таким чином, відставання аналітичних даних від АБС становить поточний "операційний день". У випадку втрати даних аналітичних завдань вони повинні бути відновлені на момент останнього закритого "операційного дня". Строк відновлення аналітичних даних залежить від того, у який момент трапився збій у системі. Якщо збій відбувся ранком, то аналітичні дані повинні бути відновлені не пізніше, ніж за 4 години. Якщо збій трапився після обіду або ввечері, то відновлення повинне бути завершене до ранку наступного дня, плюс в аналітичну

базу повинні бути завантажені дані останнього "операційного дня". Для виконання OLTP– і DSS-завдань використовується СУБД ORACLE 8i.

Аналізуючи бізнес-вимоги до інформаційної системи з наведеного приклада, виходить, що СЗД банку повинна забезпечувати роботу двох типів завдань – OLTP і DSS. Дані OLTP-завдання є критичними для системи в період з 6:00 до 23:00 (ураховується завантаження даних з АБС в аналітичну базу) і повинні забезпечувати високу готовність (простий не більше 1 години). Вимоги по надійності також високі – втрати не більше півгодини роботи. Навпроти, дані DSS-завдання не настільки критичні й вимоги по готовності не настільки високі, але повинна бути забезпечена висока надійність – втрати не припустимі.

У попередньому розділі були зазначені методи забезпечення доступності СЗД: дублювання апаратних компонентів і дублювання даних, що включає застосування різних рівнів RAID, резервне копіювання й реплікацію в резервний центр.

У наведеному прикладі ІС банку можна рекомендувати використовувати RAID 1+0 для файлів даних і журналів транзакцій OLTP-завдання, при цьому необхідно розташувати файли даних і журнали транзакцій на різних LUN. Така схема дозволить управляти продуктивністю (якщо використовуваний масив може управляти кеш-пам'яттю для окремих LUN) і забезпечить високу надійність зберігання даних. Для даних DSS-завдання рекомендується використовувати RAID 5. Цього цілком достатньо для надійного зберігання даних DSS-завдання й продуктивності при читанні даних. Для відказостійкого доступу до даних у серверах АБС необхідно встановити як мінімум по 2 FC-HBA й підключити їх до різних контролерів дискового масиву. При цьому компоненти дискового масиву повинні бути продубльовані, а ділянки кеш-пам'яті, використовувані для операцій записи, задзеркальовані й захищені від збоїв живлення.

Про вибір масиву певного класу й з певними характеристиками мова йтиме в наступному розділі. Але необхідно відзначити, що не обов'язково все компоненти масиву повинні бути дубльовані, якщо за зазначений припустимий

період простою вони можуть бути замінені, а дані при необхідності відновлені з резервних копій.

Тип додатка впливає на те, як буде здійснюватися резервне копіювання. Наприклад, для резервного копіювання СУБД ORACLE засобами Recovery Manager (RMAN) рекомендується використовувати окремий сервер (а, отже, і окремий екземпляр бази даних і дисковий простір для нього), де буде розміщений RMAN Recovery Catalog. Для резервного копіювання файлових систем цього не потрібно. Щоб відновити базу даних ORACLE необхідно мати копії журналів транзакцій, для чого рекомендується активізувати в СУБД ORACLE режим архівування журналів транзакцій (ARCHIVELOG). Для архіву журналів транзакцій буде потрібно виділити дисковий простір. Для його захисту від руйнування рівня RAID 5 буде досить. Який тип резервного копіювання використовувати (повне або інкрементальне) залежить від того, за який час можна буде скопіювати базу даних і журнали транзакцій зі стрічок на диски, і чи укладається повне резервне копіювання у відведене тимчасове вікно. Використання повного щоденного резервного копіювання дозволяє відновити базу швидше, ніж, наприклад, застосування повного щотижневого й щоденного інкрементального. При розрахунку часу відновлення СУБД ORACLE рекомендується врахувати, що дані с стрічки копіюються на диски повільніше, ніж записуються з дисків на стрічки, оскільки треба записувати метадані файлової системи [2]. Також треба врахувати, що після копіювання файлів даних і журналів транзакцій зі стрічок на диски СУБД ORACLE повинна буде виконати процедуру відновлення бази – RECOVERY, тобто "накотити" всі незавершені транзакції з журналів.

У наведеному прикладі ІС банку журнали транзакцій необхідно буде копіювати щопівгодини. Якщо зміни в базі даних АБС не великі (за день здійснюється невелике число фінансових транзакцій), то процедура RECOVERY буде виконуватися швидко (не більше десятка хвилин). Якщо база даних і журнали транзакцій можуть бути скопійовані зі стрічок на диски менш чим за 50 хвилин, досить буде робити повне резервне копіювання бази раз у добу після

закриття "операційного дня". Але якщо ці умови не виконуються, то буде потрібно використовувати більше складні технології, такі як Storage Checkpoint, реалізовані в VERITAS DBE, або засобу створення PIT-копій.

Як говоритися, збій збою ворожнеча. До цього моменту обговорювалися методи відновлення даних після "незначних" збоїв – руйнування даних у результаті відмов окремих елементів апаратури (дисків, контролерів і т.п.), помилок ПЗ або дій персоналу/користувачів системи. Збій у роботі системи може відбутися через аварію, причиною якої може бути вихід з коштуючи технічного засобу цілком (дискового масиву або сервера) або, що ще гірше, техногенна (пожежа, затоплення) або природна (землетрус, повінь) катастрофа. Від того, які вимоги пред'являються до СЗД по строках відновлення після аварій, застосовуються різні схеми резервування даних, що впливає на вибір технічних і програмних засобів і, в остаточному підсумку, на вартість рішення.

У прикладі з ІС банку, якщо потрібно забезпечити відновлення працездатності АБС протягом 1 години після техногенної катастрофи (наприклад, пожежі) у приміщенні комплексу, то, у резервному центрі необхідно крім серверів мати резервний дисковий масив, дублікати стрічок з резервними копіями й стрічковою бібліотекою достатньої потужності. Якщо обсяги даних АБС і інтенсивність їхніх змін не великі, то постійне резервне копіювання й регулярне перевезення дублікатів стрічок у резервний центр буде достатнім для захисту даних від аварії. Але у випадку більших обсягів даних (сотні гігабайт) і частих змінах у базі (великій кількості фінансових транзакцій) зазначена схема не ефективна, особливо, якщо враховувати, що резервне копіювання дає істотне додаткове навантаження на апаратний комплекс і, отже, може привести до неприпустимого зниження його продуктивності. Альтернативним рішенням у наведеному прикладі може служити передача даних між масивами, розташованими на різних площадках, що реалізується за допомогою реплікації або дзеркалювання.

Реплікація може виконуватися програмним забезпеченням, установленим на серверах (програмна реплікація), або убудованим у дискові масиви

програмним забезпеченням "прозоро" для серверів (апаратна реплікація). Для організації реплікації або дзеркалювання необхідно створити високопродуктивну інфраструктуру передачі даних, об'єднуючу обидві площадки. Як правило, для цього застосовують SAN [1].

При розробці проекту СЗД для збереження вкладених інвестицій необхідно врахувати наявні технічні засоби й найбільш оптимальним образом вписати їх у нову систему. Треба також урахувати наявність, навички й досвід обслуговуючого персоналу. Якщо кваліфікації персоналу недостатньо для обслуговування високотехнологічної системи, то крім організації навчання, ретельного пророблення регламентів збірки детальних інструкцій і іншої робочої документації, необхідно впровадити систему керування СЗД. Система керування не тільки полегшить роботу, але ще й знизить імовірність помилкових дій персоналу, які можуть привести до втрати даних.

Не секрет, що наявність плану розвитку системи полегшує її впровадження й знижує витрати на модернізацію, у порівнянні з хаотичним розвитком. З іншого боку, бізнес піддається змінам – міняються завдання, з'являються нові користувачі й т.п. Тому, навіть добре спроектована СЗД згодом перестає задовольняти бізнес-вимогам.

Допомогти в рішенні визначених проблем може знову ж впровадження все тої ж системи керування, що дозволить відстежити зміну навантажень на систему зберігання, урахувати утилізацію дискового простору, спрогнозувати потреби в його нарощуванні й т.д.

Підсумовуючи вище сказане одержуємо, що для визначення необхідних параметрів СЗД і підтримки їх у заданих рамках необхідно чітко визначити класи, режими роботи й характеристики завдань, роботу яких повинна забезпечити СЗД, необхідний обсяг зберігання даних і його можливий приріст, кількість і платформи (UNIX, Windows і ін.) серверів, що підключаються. Іншими словами, при створенні СЗД проектувальник повинен виходити від завдань і бізнес-вимог замовника.

3.2 Розробка структурної схеми

У попередньому розділі обговорювалися питання забезпечення продуктивності, доступності, масштабованості, оптимізації й експлуатації СЗД. Виходячи із цього, можна визначити, якими властивостями повинен володіти масив, щоб забезпечити рішення зазначених завдань. Вимоги наявності в масиву певних властивостей або характеристик можна розділити на категорії. Однак та саме властивість масиву може попадати в різні категорії, оскільки воно може бути використане для рішення різних завдань.

Структурна схема системи зображена на рисунку 3.1.

Приведемо перелік вимог, що часто зустрічається на практиці, але не претендує на повноту, розбитий по структурним категоріям.

Структурні вимоги:

– Ємність "сирого" (raw), тобто без розмітки на рівні RAID, дискового простору масиву повинна становити N ТБ. Якщо Вам зустрілася вимога до дискового обсягу масиву в такому формулюванні, то це означає, що планування disk layout ще не проводилося. У противному випадку формулювання було б інша: стільки-то дисків того-то обсягу й такий-то швидкості обертання, стільки-то дисків іншого обсягу й т.д.

– Число LUNs, підтримуваних дисковим масивом. Дана вимога можна чітко сформулювати знову ж тільки після планування disk layout. Але число необхідних LUN можна "грубо" порахувати по числу серверів, що підключаються до дискового масиву, з обліком виконуваних ними класів завдань. Наприклад, сервер ORACLE – 3 LUNs (дані, журнали, архів журналів), файл-сервер – 1 LUN, сервер sendmail – 2 LUNs (файли й журнал файлової системи) і т.п.

– Число серверів, що підключаються, і платформи серверів, що підключаються.

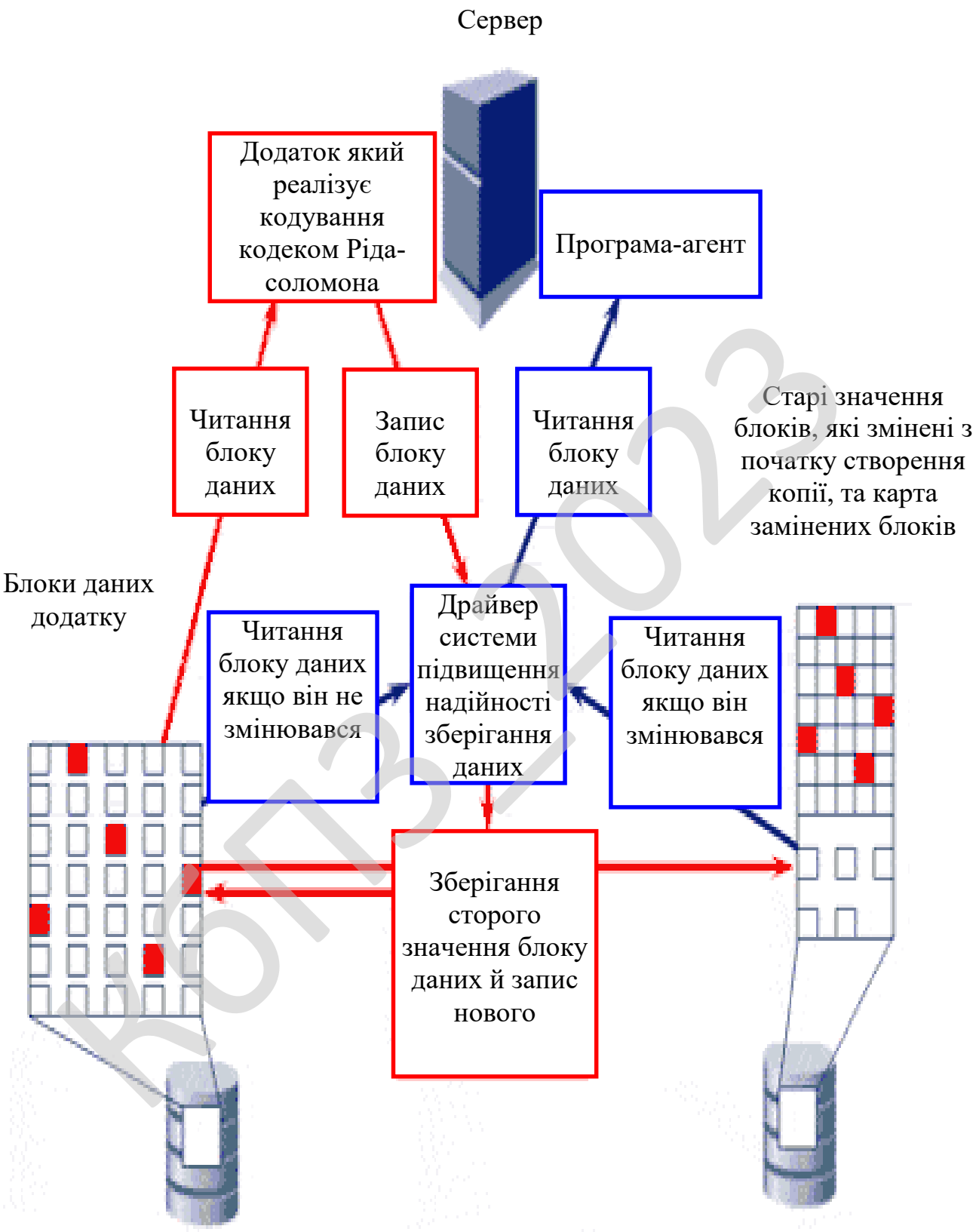

Рисунок 3.1 – Структурна схема системи

– Можливість створення PIT-копії даних засобами масиву. Дана функціональність масиву може знадобитися, якщо, наприклад, прийняте проектне рішення про завантаження даних з OLTP-завдання в DSS-завдання засобами масиву. Функція створення PIT-копій може бути реалізована різними методами – через "моментальний знімок" (SnapShot) або через повне копіювання даних (clone). Різниця між цими методами полягає в тому, що SnapShot заощаджує дисковий простір, оскільки для його створення потрібно всього лише місце для бітової карти й деякого пула для збереження старих значень змінених блоків. Навпроти, clone вимагає того ж (корисного) обсягу, що й копіюємий LUN. Однак, якщо вихідний LUN підданий частим змінам, те необхідний для підтримки SnapShot обсяг дискового простору може істотно зрости. Якщо з копією LUN, створеної за допомогою SnapShot буде вестися інтенсивна робота (велика кількість запитів на уведення-вивід), це може знизити продуктивність обміну даними з вихідним LUN. Копія LUN за допомогою SnapShot створюється моментально (звідси й назву – "моментальний знімок"), оскільки процес "копіювання" полягає тільки в створенні бітової карти. Для створення clone потрібне певний час, оскільки відбувається повне копіювання блоків даних. У цей момент навантаження по уведенню-виводу на копіюємий LUN істотно зростає. Існують проміжні способи створення PIT-копій, коли спочатку створюється SnapShot, а потім він поступово перетвориться в clone. Проектувальник повинен урахувати всі ці особливості методів створення PIT-копій і у вимогах чітко вказати який метод планується використовувати.

Вимоги до продуктивності:

– Дисковий масив повинен забезпечувати продуктивність N IOPS, а агрегована пропускна здатність масиву повинна становити M МБ/с. Як ми вже відзначали, одержати такі цифри не просто. Якщо існує прототип системи або вибір дискового масиву здійснюється для модернізації існуючої СЗД, то можна провести "натурні" виміри продуктивності й апроксимувати їх для очікуваного росту навантаження на СЗД. Якщо система створюється з "нуля", то можна

спробувати одержати ці цифри як вимоги виробника прикладного ПЗ (що практично не реально) або звернутися до виробників масивів, щоб ті провели визначення необхідних параметрів масиву (sizing). Звичайно виробники мають методики "грубої" оцінки необхідної продуктивності. Але вхідними даними для цих методик, як правило, служать очікуване число транзакцій і їх "вага" (light, medium, heavy), які теж не завжди можна визначити.

– Специфічні функції керування кеш-пам'яттю масиву. Наприклад, до таких функцій ставляться:

– можливість закріплення ділянки кеш-пам'яті за конкретним LUN (може придатися для розміщення в кеш часто використовуваних службових таблиць бази даних);

– відключення використання кеш на запис і/або читання для конкретного LUN (може знадобитися для DSS-завдань);

– забезпечення рівня сервісу у вигляді заданого рівня продуктивності (IOPS) або пропускної здатності (МБ/с) для зазначеного сервера.

Вимоги по відказостійкості й надійності зберігання даних:

– Підтримка потрібних рівнів RAID. Як правило, це рівні 1, 0+1,1+0 і 5.

– Наявність дисків "гарячої заміни" (hot-spare). Механізми використання hot-spare дисків можуть бути різні. Наприклад, можливий варіант, коли у випадку відмови диска дані з дисків порушеної RAID-групи копіюються на hot-spare диск. Але також можливий варіант, коли немає спеціально виділеного hot-spare диска – всі диски містять дані, але при цьому на всіх дисках виділена резервна область, куди копіюються дані з ушкодженою RAID-групи. Визначення необхідного методу знову ж за проектувальником.

– Захист ділянок кеш-пам'яті, що обслуговують операції запису. За винятком тих випадків, коли відключений кеш на запис, сервер одержує підтвердження завершення операції запису відразу після влучення даних у кеш-пам'ять ще до запису їх на диск. Для забезпечення цілісності даних звичайно застосовуються наступні методи:

– Дзеркалювання ділянок кеш-пам'яті, що обслуговують операції запису.

– Підтримка батареями кеш-пам'яті в плині N годин або збереження її вмісту на диски у випадку відключення зовнішнього живлення. Який із зазначених варіантів визначити у вимогах – завдання проектувальника.

– Дублювання всіх компонентів і відсутність єдиної точки відмови (SPOF). Ступінь важливості цієї вимоги залежить від режиму роботи системи й вимог до доступності сервісів. Однак, не треба забувати, що сам масив є SPOF, якщо він не задубльований іншим масивом.

– Можливість створення PIT-копій даних для використання їх у системі резервного копіювання. У ряді систем, де обробляються більші обсяги даних (терабайти), а сервіси повинні бути доступні 24x7 при більших навантаженнях, необхідно застосовувати Serverless резервне копіювання. Для цього використовується механізм створення PIT-копій засобами дискового масиву.

Вимоги по обслуговуємості:

– Можливість заміни компонентів масиву "на ходу" без зупинки системи. Виконання цієї вимоги важливо для систем, що працюють у режимі 24x7.

Вимоги по масштабованості:

– Нарощування дискового простору до N ТБ без заміни раніше встановлених дисків. Таке формулювання дозволяє "убити двох зайців" – забезпечити необхідну функціональність СЗД при росту обсягів оброблюваних даних і зберегти зроблені інвестиції. Тут може бути додана вимога: "без втрати продуктивності". Архітектура масиву може стати "вузьким місцем" і привести до того, що при черговому додаванні дисків продуктивність масиву істотно знизиться, що вплине на рівень якості сервісу.

– Розширення розміру LUN шляхом додавання нових дисків без руйнування збережених даних.Це вимога важливо не тільки для систем, що працюють у режимі 24x7, але також коли є дефіцит кваліфікованого персоналу, здатного здійснити розширення дискового простору при відсутності в масиву

даної функції. Бажано, щоб операційна система, дані якої зберігаються на розширюваному LUN, могла автоматично розширити свою файлову систему.

– Збільшення числа серверів, що підключаються, до N.

– Збільшення обсягу кеш-пам'яті до N ГБ без заміни раніше встановлених модулів.

Вимоги по керованості:

– Керування політикою використання кеш-пам'яті для різних LUN. Може знадобитися при "тонкому" налаштуванні масиву.

– Наявність засобів збору статистики про роботу масиву.

– Наявність убудованих засобів оптимізації роботи масиву. Це досить специфічна вимога, однак, наявність таких засобів може допомогти, коли буде потрібно оптимізація, а кваліфікованого персоналу, здатного її виконати, не буде.

– Інтеграція засобів управління масиву із уже розгорнутою системою керування, наприклад HP OpenView.

Щоб не порівнювати всі існуючі на ринку масиви, було б зручно розбити їх на класи. Тоді на основі отриманих вимог можна вибрати потрібний клас і вже порівнювати масиви тільки цього класу. Класи масивів придумувати не треба, вони вже визначені самим ринком, це: початковий клас (low-end), середній клас (mid-range) і вищий клас (high-end).

Масиви зазначених класів відрізняються, у першу чергу, не кількісними характеристиками, а функціональністю й архітектурою. До функціональності low-end масивів можна віднести підтримку різних рівнів RAID і можливість дублювання контролерів (якщо це не JBOD). Від масивів класу mid-range уже потрібна підтримка LUN-masking і створення PIT-копій. А в масивах класу high-end на додаток до зазначених можливостей також реалізовані апаратна реплікація, підтримка OS/390 (zOS) і керування якістю сервісу (на рівні продуктивності в IOPS або пропускної здатності в Мбайт/с).

Але все-таки основним критерієм, по якому можна віднести масив до одному із класів mid-range або high-end, є архітектура. Багато виробників

заявляють, що mid-range масиви мають модульну архітектуру, а high-end масиви – монолітну. Це не зовсім вірно. Модульна або монолітна "архітектура" говорить про конструктивний масив – збирається з окремих блоків або шаф. У дійсності архітектуру всіх mid-range масивів (і багатьох low-end) можна характеризувати як "двоконтролерну із загальною шиною".

Для high-end масивів характерна що комутирується або матрична архітектура. Очевидно, що в даній архітектурі ні "вузьких місць", тоді як в mid-range архітектурі вузькими місцями є: контролер, оскільки кожний контролер обслуговує свої RAID-групи (набір дисків, на яких реалізований один рівень RAID), шина між контролерами, обмежене число FC-AL петель до дисків, розташування дисків RAID-групи "уздовж" однієї петлі FC-AL. В high-end масивах RAID-групи розташовуються "поперек" FC-AL петель. Наприклад, в high-end масивах Hitachi RAID-група складається з 4-х або восьми дисків, де кожний диск підключається до двох різних петель від двох різних дисків контролерів. Така конфігурація дозволяє виконувати операції запису-читання із всіх дисків RAID-групи паралельно, чого не можна домогтися в mid-range масивах, коли диски однієї RAID-групи розташовані уздовж однієї петлі й доступ до них здійснюється по черзі.

Іноді ще для high-end масивів говорять про "cache-centric" архітектуру, підкреслюючи тим самим, що центральною ланкою є кеш-пам'ять, до якого мають доступ всі контролери масиву, тоді як в mid-range масивах кеш-пам'ять жорстко прив'язаний до певного контролера.

Зазначені відмінності в архітектурі приводять до втрати продуктивності при масштабуванні mid-range масивів, чого не спостерігається в high-end масивів при додаванні нових дисків. Хоча сучасні mid-range масиви мають високі характеристики масштабованості: дозволяють установлювати до двох-трьох сотень дисків, розподіляючи їх по декількох FC-AL петлях, а також нарощувати кеш-пам'ять до 8 ГБ, все-таки "вузьким місцем" залишається їхня архітектура, що є обмежником масштабованості.

Якщо дотримуватися зазначеної класифікації, то тільки масиви HDS сімейства 9900 і масиви EMC сімейства Symmetrix можна віднести до класу high-end. Масиви HP EVA і IBM ESS 800 (Shark), які позиціонуються виробниками як high-end масиви, мають архітектуру, типову для mid-range масивів.

Віртуалізація

Останнім часом у маркетинговій літературі всі частіше зустрічається поняття "віртуалізації в системах зберігання", що визначається як приховання від серверів фізичного розташування даних на дисках і подання всього дискового простору як якогось загального пула блоків.

Цей пул уже, у свою чергу, "нарізається" на логічні (віртуальні) диски (Logical Unit Number – LUN), які "видні" серверам як фізичні. У дійсності подібну організацію дискової пам'яті давно вже дозволяє створювати менеджер томів VERITAS Volume Manager. Даний тип віртуалізації одержав назву " host-based" віртуалізація.

Практично всі сучасні дискові масиви виконують функцію створення з наборів фізичних дисків логічних дисків (LUN), що одержала назва "disk array-based" віртуалізація. Це легко визначити на підставі того факту, що ряд масивів підтримують число LUNs більше, ніж фізичних дисків, як, наприклад, у недавно анонсованому масиві Sun StorEdge 3510. Питання полягає в тому, наскільки це зручно. Адміністратори воліють мати можливість управляти фізичним розміщенням файлів даних СУБД ORACLE по фізичних дисках для досягнення оптимальної продуктивності й відказостійкості. Налаштування СУБД під оптимальну продуктивність можуть стати проблематичної, якщо контролер дискового масиву не дозволяє управляти розміщенням RAID-груп на конкретних дисках.

Крім зазначених двох типів віртуалізації – " host-based" і "disk array-based", існує ще один тип – " SAN-based". У цьому типі віртуалізації приховання від серверів фізичного розташування даних здійснюється або за допомогою спеціальних пристроїв, розташованих між FC-комутаторами SAN (" in-band"

віртуалізація), або засобами самих FC-комутаторів, що зчитують інформацію про конфігурацію віртуального дискового простору із зовнішнього пристрою ("out-off-band" віртуалізація).

У цей час продуктів "SAN-based" віртуалізації на ринку мало й говорити про їхнє промислове впровадження поки не доводиться. Можливо, потреба в цьому типі віртуалізації з'являться тоді, коли обсяги даних підприємств зростуть настільки, що для їхнього оперативного зберігання не буде вистачати декількох high-end дискових масивів.

3.3 Розробка функціональної схеми

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

– сам носій інформації – сервер у мережі;

– кодер Ріда-Соломона, який використовується, коли відбувається запис інформації на сервер, при цьому необхідно враховувати, що об’єм інформації, яка записується на диск серверу, повинна бути меншою, ніж об’єм диску, в зв’язку, з тим, що коди Ріда-Соломона відносяться до кодів з надмірністю, за рахунок якої й відбувається кодування;

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

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

– дані, які потрібно зберегти на сервері;

– поліном для кодування;

– відмовостійки коди.

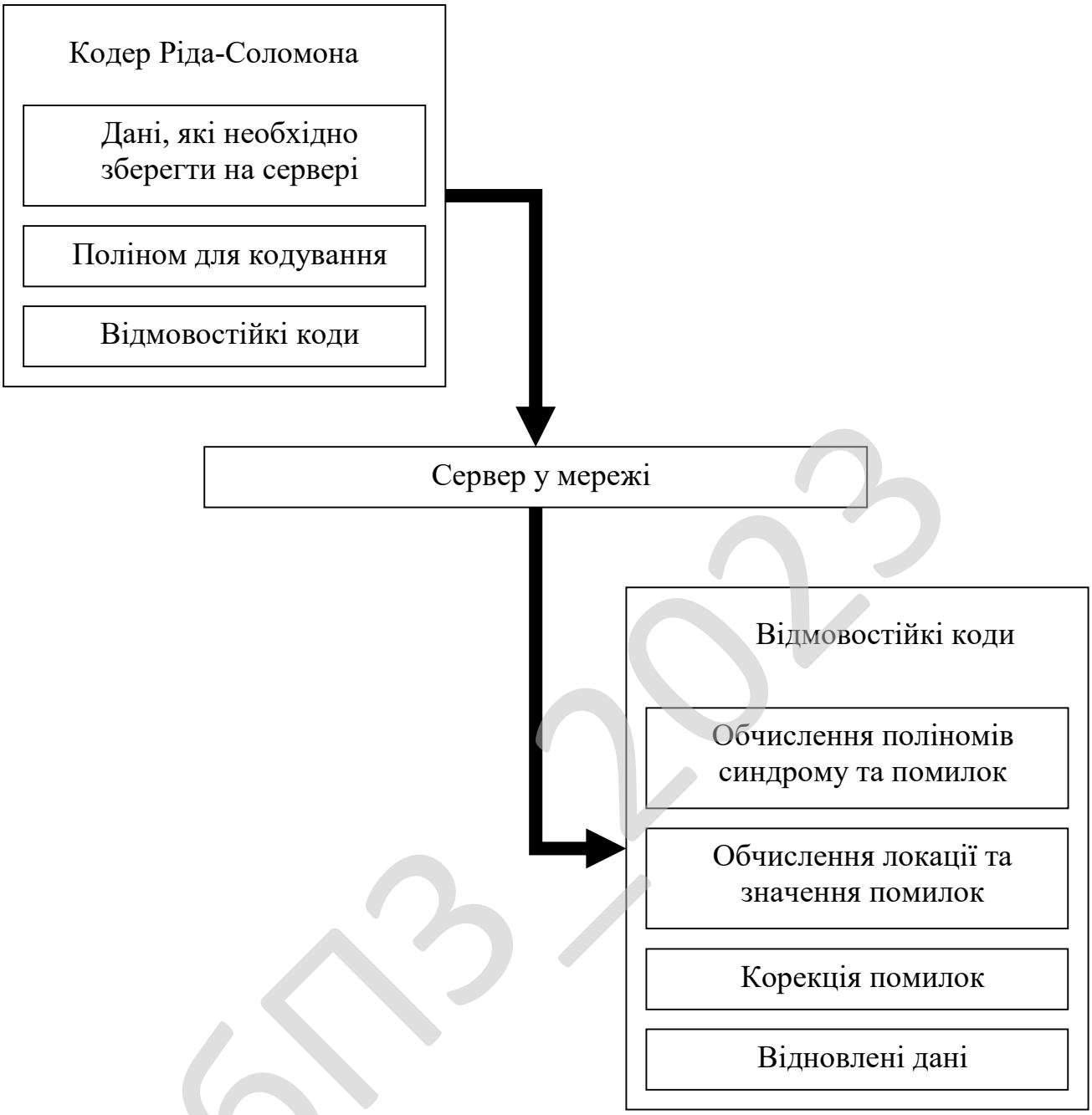

Рисунок 3.2 – Функціональна схема системи

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

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

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

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

Цей функціональний блок містить у собі наступні блоки:

– відмовостійкий код;

– блок обчислення поліномів синдрому та помилок;

– блок обчислення локації та значень помилок;

– блок корекції помилок;

– відновлені за допомогою коді Ріда-Соломона дані.

Декодер працює наступним чином.

Введемо позначення:

– r(x) – Отримане кодове слово.

– Si – Синдроми.

– L(x) – Поліном локації помилок.

– Xi – Положення помилок.

– Yi – Значення помилок.

– c(x) – Відновлене кодове слово.

– v – Число помилок.

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

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

Обчислення синдрому

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

Знаходження позицій символьних помилок

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

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

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

Знаходження значень символьних помилок

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

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

Код Ріда-Соломона

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

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

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

Код Ріда-Соломона був винайдений в 1960 році співробітниками лабораторії Лінкольна Массачуссетского технологічного інституту Ірвином Рідом і Густавом Соломоном. Ідея використання цього коду була представлена в статті «Polynomial Codes over Certain Finite Fields». Перше застосування код Ріда-Соломона одержав в 1982 році в серійному випуску компакт-дисків. Ефективний алгоритм декодування був запропонований в 1969 році Елвином Берлекемпом і Джеймсом Месси.

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

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

Поліном, що породжує

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

Теорема 3.1

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

Теорема 3.2

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

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

Матриця, що породжує

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

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

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

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

Матрицю G можна записати в символьній формі:

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

Перевірочна матриця

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

Тоді:

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

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

Тоді нормований поліном $g(x)$ мінімального ступеня над полем $GF(q)$, корінням якого є d-1 ступенів, що йдуть підряд, $\alpha^{i_0}, \alpha^{i_0+1}, ..., \alpha^{i_0+d-1}$ елемента α,

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

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

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

Властивості

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

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

Виправлення багаторазових помилок

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

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

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

Практична реалізація

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

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

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

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

Кодування

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

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

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

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

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

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

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

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

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

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

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

Символу «С» відповідають дві константи $GF(n)$.

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

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

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

Декодування

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

– Обчислює синдром помилки.

– Будує поліном помилки.

– Знаходить корінь даного полінома.

– Визначає характер помилки.

– Виправляє помилки.

Обчислення синдрому помилки

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

Побудова полінома помилки

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

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

Алгоритм Евкліда виконується наступним чином.

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

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

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

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

Коректність цього алгоритму випливає з наступних двох тверджень:

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

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

Знаходження корня

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

Визначення характеру помилки і її виправлення

По синдрому помилки й знайдених корінь полінома за допомогою алгоритму Форни визначається характер помилки й будується маска перекручених символів.

Ця маска накладається на кодове слово за допомогою операції XOR і перекручені символи відновлюються.

Після цього відкидаються перевірочні символи й виходить відновлене інформаційне слово.

Застосування

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

Код Ріда-Соломона використовується при записі й читанні в контролерах оперативної пам'яті, при архівуванні даних, запису інформації на жорсткі диски (ECC), запису на CD/DVD диски.

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

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

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

Надмірність приблизно 25% від вихідної інформації.

При записі на цифрові аудіокомпакт-диски (Compact Disc Digital Audio –

CD-DA) використовується стандарт Red Book.

Корекція помилок відбувається на двох рівнях C1 і C2.

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

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

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

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

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

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

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання магістерського проектування, наведена на рисунку 3.3.

Після початку роботи на екран виводиться головне вікно ПЗ з якого можна перейти до обрання конфігурації, моніторингу томів на резервних дисках та формування списку файлів та у подальшому перейти до вибору резервних дисків, створення томів для відновлення даних та записом томів на резервні диски. Чи через вибір дисків для запису відновленої інформації та декодування підключити

шифруючий фільтр та провести перевірку цілісності даних з відновленням пошкоджених томів та записом даних на вказані диски.

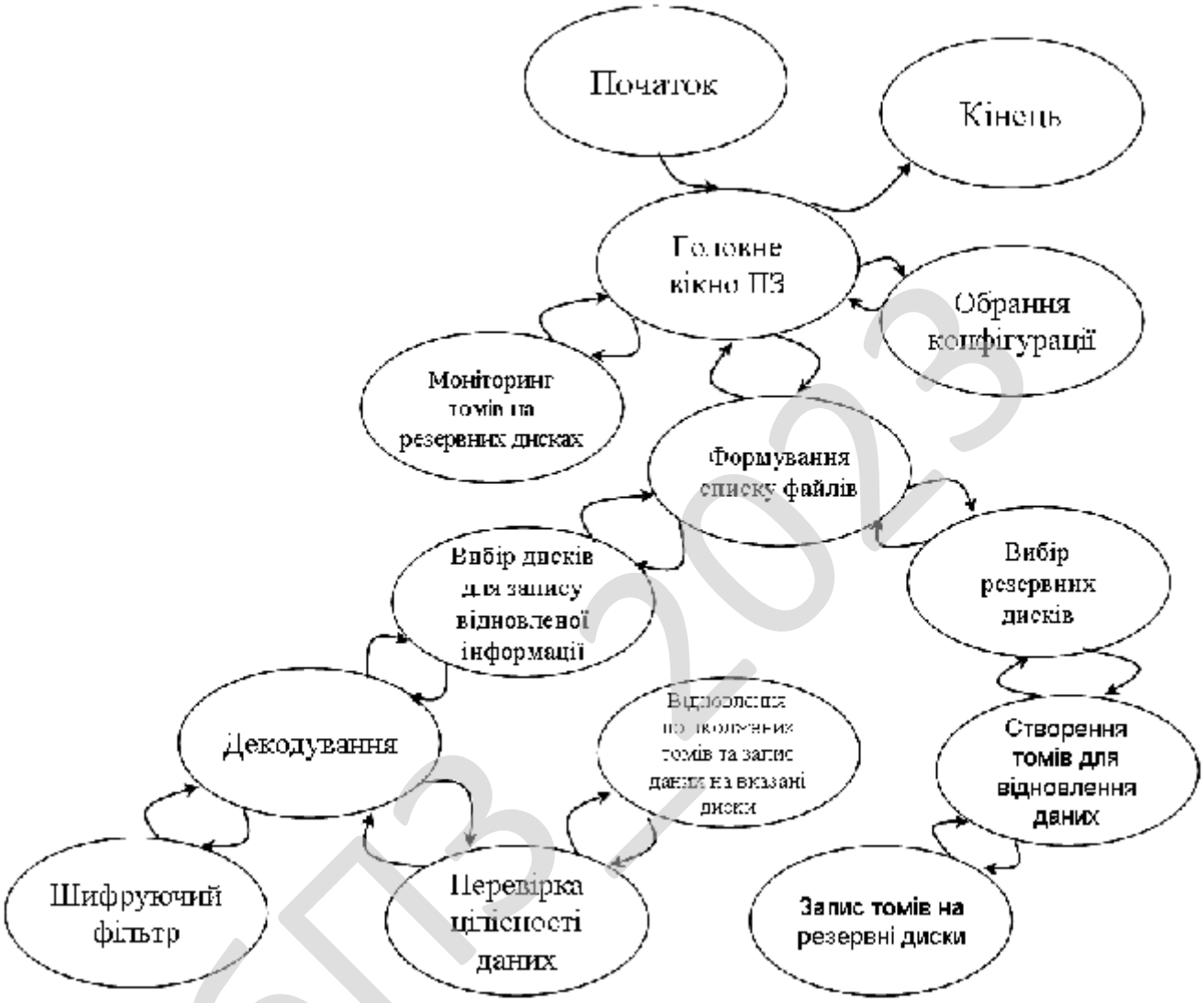

Рисунок 3.3 – Діаграма взаємодії процесів

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків:

– Відновлення пошкоджених томів та запис даних на вказані диски.

– Сканування та виведення списку томів диску.

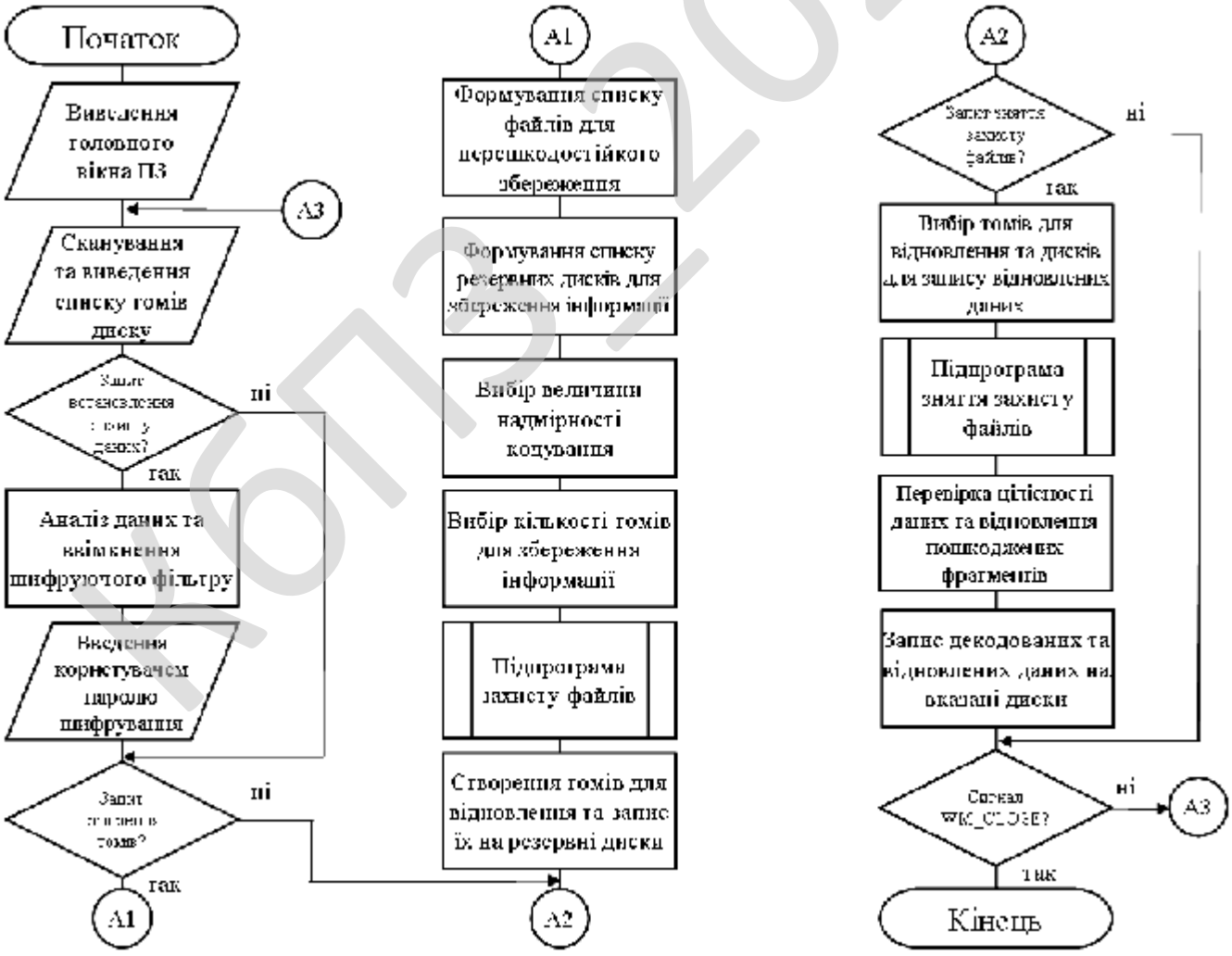

Рисунок 4.1 – Блок-схема основної програми

– Запит встановлення захисту даних?

– Аналіз даних та ввімкнення шифруючого фільтру.

– Введення користувачем паролю шифрування.

– Запит створення томів?

– Формування списку файлів для перешкодостійкого збереження.

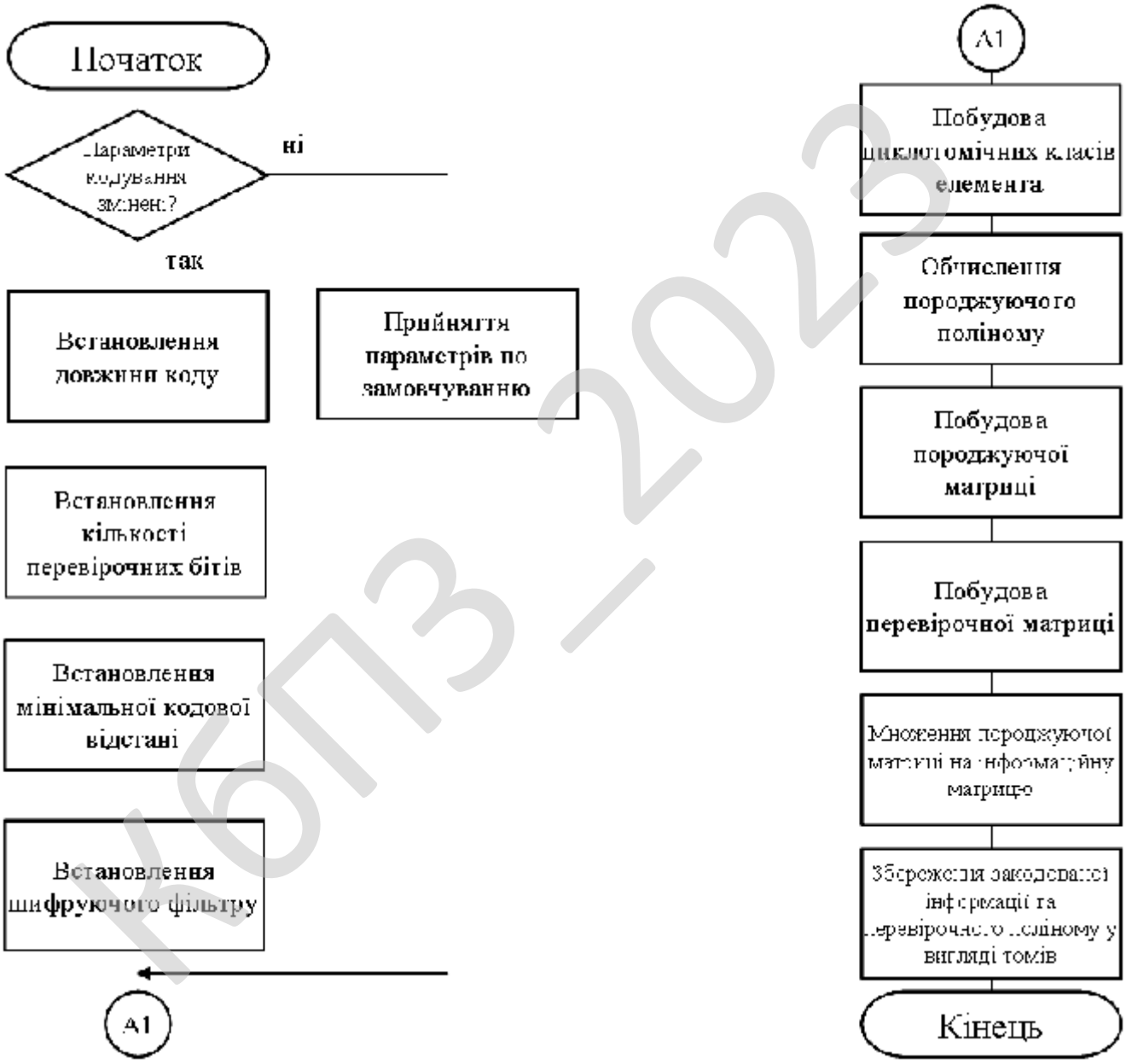

Рисунок 4.2 – Блок-схема підпрограми захисту файлів

– Формування списку резервних дисків для збереження інформації.

– Вибір величини надмірності кодування.

– Вибір кількості томів для збереження інформації.

– Підпрограма захисту файлів.

– Створення томів для відновлення та запис їх на резервні диски.

– Запит зняття захисту файлів?

– Вибір томів для відновлення та дисків для запису відновлених даних.

– Підпрограма зняття захисту файлів.

– Перевірка цілісності даних та відновлення пошкоджених фрагментів.

– Запис декодованих та відновлених даних на вказані диски.

– Сигнал WM_CLOSE (запит).

На рисунку 4.2 наведено блок-схему підпрограми захисту файлів. Її робота складається з виконання наступних кроків:

– Параметри кодування змінені (запит).

– Встановлення довжини коду.

– Встановлення кількості перевірочних бітів.

– Встановлення мінімальної кодової відстані.

– Встановлення шифруючого фільтру.

– Побудова циклотомічних класів елемента.

– Обчислення породжуючого поліному.

– Побудова породжуючої матриці.

– Побудова перевірочної матриці.

– Множення породжуючої матриці на інформаційну матрицю.

– Збереження закодованої інформації та перевірочного поліному у вигляді томів.

Опис алгоритмів функціонування системи

Коди Ріда – Соломона (Reed-Solomon codes) – недвійкові циклічні коди, що дозволяють виправляти помилки в блоках даних. Елементами кодового

вектора є не біти, а групи бітів (блоки). Дуже поширені коди Ріда – Соломона, що працюють з байтами (октет).

Код Ріда – Соломона є окремим випадком БЧХ-коду. Коди Боуза – Чоудхурі – Хоквінгема (БЧХ-коди) – в теорії кодування це широкий клас циклічних кодів, що застосовуються для захисту інформації від помилок (попередня корекція помилок).

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

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

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

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

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

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

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

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

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

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

```
/*-------------------------------------
* згенероване кодове слово описується наступною формулою:
* C(x) = data(x) * x ^ (n - k) + b(x), де ^ означає зведення в ступінь
-------------------------------------*/
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 – шуканий залишок
                        for (j = n - k - 1; j > 0; j-j--) b[j] = b[j - 1]; b[0] = 0;
                }
        }
}
```

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

Потім, використовуючи ітеративний алгоритм Берлекэмпа (Berlekamp), ми знаходимо поліном локатора помилки – elp[i]. Якщо ступінь elp перевищує собою величину t, ми неспроможні скорегувати всі помилки й обмежуємося виводом повідомлення про непереборну помилку, після чого робимо аварійний вихід з декодера. Якщо ж ступінь elp не перевищує t, ми підставляємо alpha**i, де i = 1...n в elp для обчислення корінь полінома. Обіг знайдений корінь дає нам позиції перекручених символів. Якщо кількість певних позицій перекручених символів менше ступеня elp, перекручуванню піддалося більш ніж t символів і ми не можемо відновити їх.

У всіх інших випадках відновлення оригінального вмісту перекручених символів цілком можливо. У випадку, коли кількість помилок свідомо велико, для їхнього виправлення декодуємі символи проходять крізь декодер без яких або

змін.

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

```
// обчислюємо поліном локатора помилки через ітеративний алгоритм
// Берлекемпа. Випливаючи термінології Lin and Costello (див. "Error
// Control Coding: Fundamentals and Applications" Prentice Hall 1983
// ISBN 013283796) d[u] являє собою m ("мю"), що виражає
// розбіжність (discrepancy), де u = m + 1 і m є номер кроку
// з діапазону від -1 до 2t. У Блейхута та ж сама величина
// позначається D(x) ("дельта") і називається нев'язання.
// l[u] являє собою ступінь elp для даного кроку ітерації,
// u_l[u] являє собою різницю між номером кроку й ступенем elp
            // ініціалізація елементи таблиці
            d[0] = 0; // індексна форма
            d[1] = s[1]; // індексна форма
            elp[0][0] = 0; // індексна форма
            elp[1][0] = 1; // поліноміальна форма
            for (i = 1; i < n - k; i++)
            {
                  elp[0][i] = -1; // індексна форма
                  elp[1][i] = 0; // поліноміальна форма
            }
            l[0] = 0; l[1] = 0; u_lu[0] = -1; u_lu[1] = 0; u = 0;
            do
            {
                  u++;
                  if (d[u] == -1)
                  {
                        l[u + 1] = l[u];
                        for (i = 0; i <= l[u]; i++)
                        {
                              elp[u + 1][i] = elp[u][i];
                              elp[u][i] = index_of[elp[u][i]];
                        }
                  }
                        else
                  {
                        // пошук слів з найбільшим u_lu[q], таких що d[q] != 0
                        q = u - 1;
                        while ((d[q] == -1) && (q > 0)) q-q--;
                        // знайдений перший ненульовий d[q]
                        if (q > 0)
                        {
                              j = q;
                              do
```

```
                        {
                                j-j-- ;
                                if ((d[j] != -1) && (u_lu[q] < u_lu[j]))
                                q = j;
                        } while (j > 0);
                };
 // як тільки ми знайдемо q, такий що d[u] !=0
// i u_lu[q] є максимум
// запишемо ступінь нового elp полінома
                if (l[u] > l[q] + u - q) l[u + 1] = l[u]; else l[u + 1] =
l[q] + u - q;
// формуємо новий elp(x)
                for (i = 0; i < n - k; i++) elp[u + 1][i] = 0;
                for (i = 0; i <= l[q]; i++)
                        if (elp[q][i] != -1)
                                elp[u + 1][i + u - q] = alpha_to[(d[u] + n -
 d[q] + elp[q][i]) % n];
                for (i = 0; i <= l[u]; i++)
                {
                        elp[u + 1][i] ^= elp[u][i];
                        // перетворимо старий elp
                        // в індексну форму
                        elp[u][i] = index_of[elp[u][i]];
                }
            }
            u_lu[u + 1] = u - l[u + 1];
 // формуємо (u + 1)'ю нев'язання
//---------------------------
            if (u < n - k)          // на останній ітерації розбіжність
            {                       // не було виявлено
           if (s[u + 1] != -1) d[u + 1] = alpha_to[s[u + 1]]; else d[u + 1] = 0;
                for (i = 1; i <= l[u + 1]; i++)
                        if ((s[u + 1 - i] != -1) && (elp[u + 1][i] != 0))
                        d[u + 1] ^= alpha_to[(s[u + 1 - i] + index_of[elp[u +
1][i]]) % n];
                // переводимо d[u + 1] в індексну форму
                d[u + 1] = index_of[d[u + 1]];
            }
      } while ((u < n - k) && (l[u + 1] <= t));
      // розрахунок локатора завершений
      //------------------------
      u++ ;
```

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

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

```
        // переводимо recd[] у поліноміальну форму
        for (i = 0; i < n; i++)
                if (recd[i] != -1)
                        recd[i] = alpha_to[recd[i]];
                else
                        recd[i] = 0 ; // виводимо інформаційне слово як є
        }
                else                        // помилок не виявлено
        for (i = 0; i < n; i++) if (recd[i] != -1) recd[i] = alpha_to[recd[i]];
else recd[i] = 0;
}
```

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм MMB, в основі якого лежить змішування операцій різних алгебраїчних груп. MMB – ітеративний алгоритм, що складається з лінійних дій (XOR і використання ключа) і паралельного застосування чотирьох великих оборотних нелінійних підстановок. Ці підстановки визначаються за допомогою множення по модулю 2^{32}-1 з постійними множниками. У підсумку з'являється алгоритм, що використовує 128-бітовий ключ і 128-бітовий блок.

Алгоритм MMB оперує 32-бітовими підблоками тексту (x_0, x_1, x_2, x_3) і 32-бітовими підблоками ключу (k_0, k_1, k_2, k_3). Це спрощує реалізацію алгоритму на сучасних 64-бітових процесорах. Чергуючись із операцією XOR, шість разів використовується нелінійна функція *f*. Запишемо операції алгоритму (всі операції з індексами виконуються по модулю 4):

$$x_i = x_i \oplus k_i \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+1} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+2} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_i \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+1} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+2} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

Функція f виконується в три кроки:

1. $x_i = c_i * x_i$ для $i = 0..3$ (Якщо на вході множення одні одиниці, то на виході – теж одні одиниці).

2. Якщо молодший значущий біт $x_0 = 1$, то $x_0 = x_0 \oplus C$. Якщо молодший значущий байт $x_3 = 0$, то $x_3 = x_3 \oplus C$.

3. $x_i = x_{i-1} \oplus x_i \oplus x_{i+1}$ для $i = 0..3$.

Всі операції з індексами виконуються по модулю 4. Операція множення на кроці 1 виконується по модулі 2^{32}-1. Спеціальний випадок для даного алгоритму: якщо другий операнд дорівнює 2^{32}-1, результат теж дорівнює 2^{32}-1. В алгоритмі використовуються наступні константи:

$$C = 2aaaaaaa,\ c_0 = 025f1cdb,\ c_1 = 2*c_0,\ c_2 = 2^3 * c_0,\ c_3 = 2^7 * c_0.$$

Константа C – «найпростіша» константа без кругової симетрії, високою трійковою вагою й нульовим молодшим значущим бітом. У константи c_0 є інші особливі характеристики. Константи c_1, c_2 і c_3 – зрушені версії c_0, і служать для запобігання атак, заснованих на симетрії.

Розшифрування виконується у зворотному порядку, Етапи 2 і 3 інверсні їм самим. На етапі 1 замість c_i використовується c_i^{-1}. Значення c_0^{-1} = 0dad4694.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено головне вікно програми. З нього видно, що інтерфейс користувача програми складається з таких логічних блоків:

– Меню: Додатки; Налаштування; Довідка.

– Файли. Навігатор для обрання файлів резервування.

– Резервні копії даних. Створені резервні дані на додаткових носіях.

– Дії (функції ПЗ): Збереження даних; Відновлення даних; Сканування; Пошук помилок; Виправлення помилок; Налаштування; Авторське право.

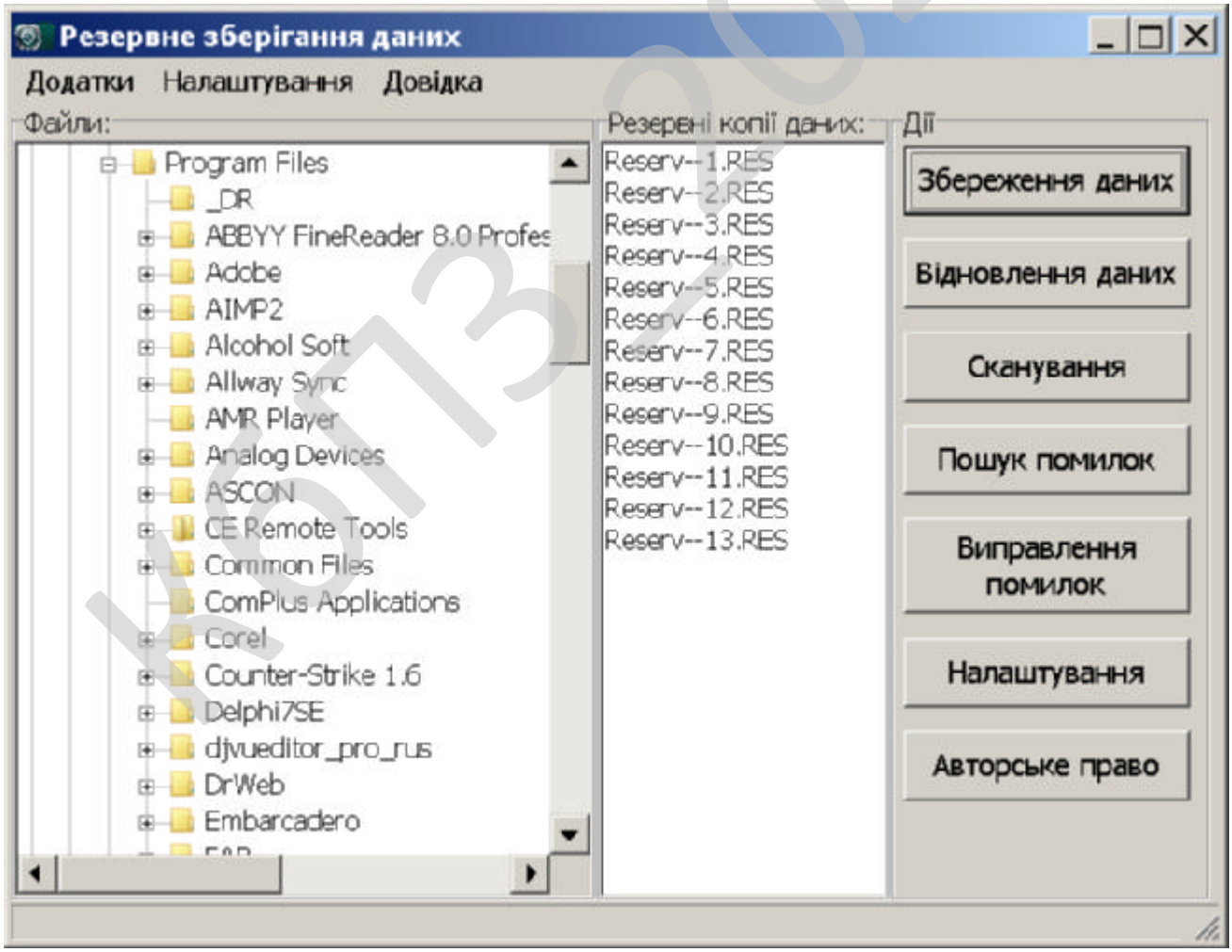

Рисунок 5.1 – Головне вікно програми

На рисунку 5.2 зображено авторське право. Авторське право є ключовою галуззю права інтелектуальної власності; воно призначене захищати лише зовнішню форму вираження об'єкта, тобто їхнє матеріальне втілення. Авторське право не може використовуватись для захисту абстрактних ідей, концепцій, фактів, стилів та технік, що можуть бути використані у творі. Захист авторського права — одна з важливих категорій теорії цивільного та цивільно-процесуального права. Під захистом авторських прав слід розуміти передбачені законом заходи із їхнього визнання, припинення їхнього порушення, застосування до правопорушників заходів юридичної відповідальності. Захист особистих немайнових і майнових прав суб'єктів авторського права здійснюється в порядку, встановленому адміністративним, цивільним і кримінальним законодавством. Було обрано Shareware умову розповсюдження. Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (неповнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

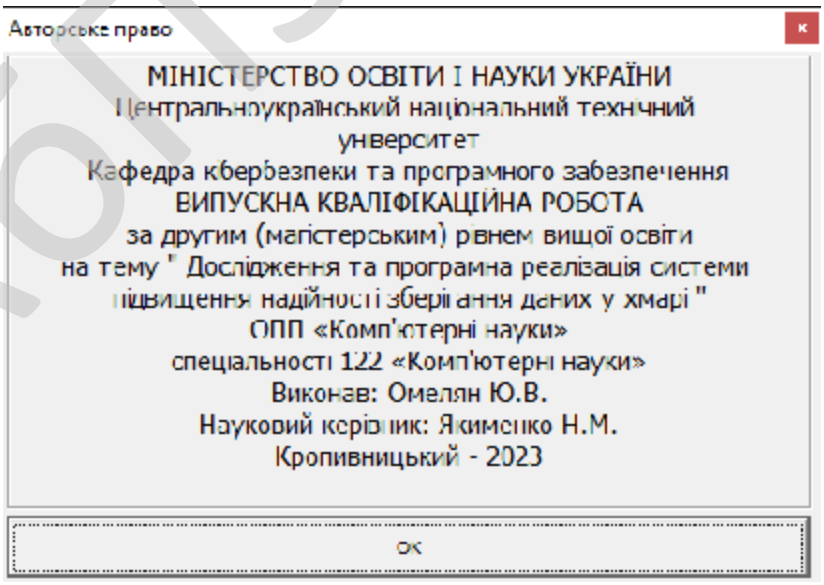

Рисунок 5.2 – Довідка автора

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи підвищення надійності зберігання даних у хмарі.

Метою розробки є дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі.

Об'єктом дослідження є процес підвищення надійності зберігання даних у хмарі.

Предметом дослідження є методи підвищення надійності зберігання даних у хмарі.

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

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

– Удосконалено метод підвищення надійності зберігання даних у хмарі.

– Розроблено вітчизняний продукт підвищення надійності зберігання даних у хмарі, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

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 магістерській роботі було проведене дослідження та виконана програмна реалізація системи підвищення надійності зберігання даних у хмарі.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім 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	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт	N	1
2. Кількість екземплярів програм, шт	Ne	50
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
7. Кількість макетів вхідної інформації	–	3

Продовження табл. 7.1

1	2	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 табл. 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	–	50000
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	55
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.:

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

де A – коефіцієнт Боема, A=2,45; Size – загальний об'єм відлагодженого програмного коду, тис. рядків; B – показник ступеня, що визначається співвідношенням

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

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

$$B=1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}026$$

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

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

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

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

$$T_{уточн}= 6{,}78\cdot(0{,}88\cdot0{,}93\cdot0{,}88\cdot0{,}91\cdot0{,}95\cdot1\cdot1\cdot0{,}87\cdot1{,}22\cdot1{,}16\cdot1{,}1\cdot1{,}1\cdot$$
$$\cdot1{,}12\cdot1{,}1\cdot1\cdot1{,}1\cdot1{,}1)=9{,}37 \text{ люд-міс.}$$

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:

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

де C – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0{,}3\cdot2{,}66\cdot9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)}\cdot91=153 \text{ люд/день}$$

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	153	Ф 7.1-7.4
Впровадження	13	Д13
Всього	194	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

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

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

де Fpq – плановий фонд робочого часу одного спеціаліста, днів, Тпз – трудомісткість розробки програмного забезпечення люд-дні,

$$Ч = \frac{194 \cdot 1}{60\text{-}5} = 3,5 \text{ ставки}$$

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.
Системний блок ПК	90	10	900	15
Монітор	60	10	600	10
Клавіатура	30	10	300	5
Маніпулятор «мишка»	30	10	300	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	3	90	1,5
Кабельні господарства ЛОМ на 1 м.п.	2,5	250	625	10,42
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	52,58

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

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

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

$$Ф_{др}^{с} = \frac{53 \cdot 3}{1,2} = 132,5 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

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

$Ч_{ел}$=132,5/(60·8)=0,28 ставки

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

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.

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

Посада	Вид роботи	Час	Кількість штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, серверу доступу АДСЛ (ОС Linux), Wi-Fi налаштування ADSL, VPN, PPPoE, Frame Relay	0,5	0,25
	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 д мережі Інтернет	0,5	
Всього		2	

Продовження таблиці 7.4

Посада	Вид роботи	Час	Кількість 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	
	Оформлення банерів і промо-сторінок	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
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	

Складемо штатний розклад виконавців:

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (ІТ-менеджер)	1	16000	48000
Продакт-менеджер	0,25	14000	10500
Інженер-програміст	3,5	15000	157500
Інженер-електронщик	0,28	12000	10080
Інженер-системотехнік	0,25	12000	9000
Адміністратор мережі	0,25	13000	9750
Системний програміст	0,25	13000	9750
Дизайнер WEB	0,25	15000	11250
Інженер-верстальник	0,25	14000	10500
Бухгалтер-економіст	0,5	15000	22500
Всього за період розробки	$R_{сп}$=6,78	-	$Ф_{роб}$=298830

Розрахуємо середньоденну зарплату одного виконавця:

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

де Фроб – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{298830}{6,78 \cdot 60} = 734 \ грн.$$

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

$$Б_{уд} = R_{сп}^{1} S_{у} Ц_{пл}, \quad (7.9)$$

де $R_{сп}^{1}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць. Sу – 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

$$I_{нв} = R_{сп}^{1} \cdot Ц_{м}, \quad (7.10)$$

де Цм – ціна меблів для одного робочого місця, грн.

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

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 по прайсу Інтернет магазину Компбест за 16.10.23 – джерело https://compbest.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		10947
Системний блок		7347
Процесор	Socket 1151 Intel Core i3-6100 (2 (4) ядра по 3.7 GHz) BOX	-
Системна плата	FUJITSU Mainboard D3402-B μATX. 5 x Serial ATA III 600 Interface (up to 6GBit/s, NCQ, AHCI, RAID 0/1/5/10) 1 x M.2 Socket (2260/2280) for PCIe (4 lanes) or SATA based M.2 modules, AHCI Boot support	-
Відеокарта	Вбудована в процесор Intel HD Graphics 530	-
Жорсткий диск	1000 GB HDD + SSD M.2 2280 250GB ADATA	-
Оперативна пам’ять	8 GB DDR4 (2 модулі по 4 GB)	-
DVD-привод	DVD -RW/+RW , LG SATA SuperMulti Bulk 22x, SecurDisc, black	-
Корпус	Terra Tower, 400W (120mm big fan), full-ATX, БЖ 4xSATA, Air Duct, 2xUSB 3.0, Mic+Audio, silver/black	-
Кардрідер внутрішній	USB 3.0 Card reader STORM CR, int. 3.5", 2*USB3.0+AUDIO+1394, multi: All Type Cards, black	-
інше	Клавіатура, мишка, DISPLAY PORT TO HDMI 1.0M CABLEXPERT	-

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Монітор	LG W2363V-WF Wide LCD 2ms, 70 000:1, 300кд/м2, 170/160, D-Sub / Glossy White	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струменевий	Epson Stylus Photo P50 (C11CA45341) + USI cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

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 7.7 – 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	8	10947	8757,6	96333,6
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	114885,1

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	114885	-	-
Всього по групі	114885	50	57442,5
Група 5			
4. Вимірювальні пристрої	5190	-	-
5. Господарський інвентар	28000	-	-
Всього по групі	33190	25	8297,5
Нематеріальні активи			
6. Нематеріальні активи	50000	10	5000
Разом	$К_р$= 1606075		$А_р$= 141140

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців

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

де Ne – Кількість екземплярів програм, шт.

$$З_о = 734 \cdot 194/50 = 2848 \text{ грн}$$

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, %

$$З_д = 2848 \cdot 10 \cdot 0{,}01 = 285 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=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, %

$$С_{оц} = 0{,}01 \cdot 22(2848+285) = 689 \text{ грн}$$

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

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

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 2848 \cdot 15 \cdot 0{,}01 = 427 \text{ грн}$$

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 $n_{міс}$ приймаємо 2 пачки паперу на місяць розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=220 грн., визначаємо вартість паперу за період розробки $N_м$ =3 міс:

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

$$З_{М1} = 220 \cdot 3 \cdot 2 = 1320 \text{ грн}.$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 50):

$$З_{М2} = \sum Ц_{д}., \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 25 грн./шт., DVD-R box – 35 грн./шт.

$$З_{М2} = 50 \cdot 25 + 35 = 1285 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_{з}., \quad (7.18)$$

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.

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

$$З_{М} = (1320 + 1285 + 1702)/50=86 \text{ грн.}$$

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

$$О_{п} = З_{о} \cdot Н_{п} \cdot 0{,}01, \quad (7.19)$$

де Нп – норматив витрат на освоєння нових мов програмування, %

$$О_{п} = 2848 \cdot 15 \cdot 0{,}01=427 \text{ грн}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 50 прим.)

$$А_{м} = \frac{А_{р} \cdot N_{міс}}{N_{е} \cdot 12}, \quad (7.20)$$

де Ар – загальна річна сума амортизаційних відрахувань, грн.

$$А_{м} = 141140 \cdot 3/(50 \cdot 12)=706 \text{ грн}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

$$С_{п} = З_{о} + З_{д} + С_{оц} + Г_{осп} + З_{м} + О_{п} + А_{м}. \quad (7.21)$$

$$C_n = 2848+285+689+427+86+427+706 = 5468 \text{ грн.}$$

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 55%

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

де Рс – рівень рентабельності, %

$$П_p = 0{,}01 \cdot 55 \cdot 5468 = 3007 \text{ грн.}$$

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	$З_о$	2848
2. Додаткова зарплата виконавців	$З_д$	285
3. Відрахування на соціальні потреби	$С_{оц}$	689
4. Загальногосподарські витрати	$Г_{осп}$	427
5. Витрати на матеріали	$З_М$	86
6. Освоєння нових операційних систем, мов програмування	$О_п$	427
7. Амортизація основних фондів	$А_м$	706
8. Повна собівартість програмного забезпечення	$С_п$	5468
9. Плановий прибуток	$П_р$	3007
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	8475
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	1695
12.Відпускна ціна програмної продукції Ц = Цп + ПДВ	$Ц$	11482

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 7.10.

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

Найменування капітальних вкладень	Сума за варіантами, грн	
	Базовий	Новий
Вартість програмної продукції	–	11482
Всього капітальних витрат	–	11482

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.

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	$З_р$	122390	50996
2. Витрати на електроенергію	$З_{ел}$	1045	261
3. Витрати на амортизацію	$З_{ам}$	0	2871
Всього витрат за рік	$І$	123435	54128

Витрати на оплату праці:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \quad (7.23)$$

де Тр – кількість годин обслуговування системи за рік, *год.*; Зг – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення кількість годин на реалізацію поставленої задачі зменшилась з 600 годин до 250 годин на рік, тому витрати на обслуговування склали:

$$З_{р\,баз} = 600 \cdot 152 \cdot 1{,}1 \cdot 1{,}22 = 122390 \; грн.$$

до

$$З_{р\,нов} = 250 \cdot 152 \cdot 1{,}1 \cdot 1{,}22 = 50996 \; грн.$$

Витрати на електроенергію визначаються з урахуванням спожитої потужності (Пел) в кіловатах, часу експлуатації технічних засобів (Тр) в годинах та ціни однієї кіловат-години (Цел).

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

Зел баз=·0,475·1000·2,2 = 1045 грн

Зел нов=·0,475·250·2,2 = 261 грн

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	–	11482	–	2870,5
Всього відрахувань	-	–	11482	–	2870,5

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

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

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

$$Ев = (8475 - 5468) \cdot 50 - (0,05 \cdot 1408000 + 0,5 \cdot 114885 +$$
$$+0,25 \cdot 33190 + 0,1 \cdot 50000) \cdot 3/12 = 115065 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника.

$$T_в = \frac{1606075}{(8475 - 5468) \cdot 50 \cdot 12 / 3} = 2,7 \text{ років}.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (123435 - 54128) - 0,25 \cdot 11482 = 66437 \text{ грн.}$$

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

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

$$T_{сп} = \frac{11482}{123435 - 54128} = 0{,}17 \ року$$

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.	50
2. Повна собівартість розробленої програми	Грн.	5468
3. Ціна розробленої програми	Грн.	8475
4. Плановий прибуток від реалізації розробленої програми	Грн.	3007
5. Рентабельність програмної продукції	%	55
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1606075
7. Загальний прибуток від реалізації програмної продукції	Грн.	150350
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	115065
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	2,7
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	11482
11.Величина економічного ефекту у користувача програмної продукції	Грн.	66437
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,17

7.9 Висновки

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

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Охорона здоров'я працівників, забезпечення безпеки умов праці, ліквідація професійних захворювань і виробничого травматизму повинна складати одну з головних завдань роботодавця.

Основою охорони праці є науковий аналіз умов праці, технологічних процесів, виробничого обладнання, робочих місць, трудових операцій, організації виробництва з метою виявлення шкідливих і небезпечних виробничих факторів, їх властивостей, особливостей виливу на організм людини. На підставі такого аналізу розробляються заходи та засоби, спрямовані на мінімізацію несприятливого впливу виробничих факторів, створення безпечних та нешкідливих умов праці.

Для того, щоб об’єктивно проаналізувати відповідність умов праці діючим нормативно-правовим актам, необхідно здійснити санітарно-гігієнічну характеристику умов праці відділу, в якому працює програміст, над розробкою даного програмного продукту.

В зв’язку з цим необхідно сконцентрувати увагу на небезпечних і шкідливих чинниках пов’язаних з постійною роботою за комп’ютером.

Електробезпека є одним із критичних питань для співробітників, що працюють із технікою, яка одержує живлення з електричної мережі. При невиконанні норм електробезпеки можлива поразка електричним струмом.

8.2 Аналіз умов праці програміста

Умови праці в приміщенні, в якому знаходиться робоче місце програміста є сприятливими. Приміщення обладнане автономною системою газового опалення, основною перевагою якого є програмування режиму роботи в залежності від

погодних умов, оскільки клімат є нестійким. Використовується система природної та штучної вентиляції, що забезпечує ефективну циркуляцію повітря. В кабінеті знаходяться 1 кондиціонер (CARRIER 42QCR018/38QCR018).

Засоби копіювальної техніки знаходяться на достатньо далекій відстані від робочих місць, оскільки приміщення складає 20 м^2, а у відділі налічується два працівники, тобто концентрація озону та оксиду азоту в повітрі є невисокою. Таким чином на кожного програміста приходиться 10,0 м^2 що відповідає нормам Державним санітарним правилам і нормам ДСанПіН 3.3.2.007-98 [1]. Висота стелі приміщення складає 3 метри, що також не порушує нормативні вимоги.. Прибиральники підтримують порядок в службових приміщеннях, дотримуються санітарно-гігієнічних норм по прибиранню приміщень, витирають пил, підмітають підлогу наприкінці кожного робочого дня.

В цілому потрібно відмітати застаріліть офісної техніки та відсутність клавіатур з ергономічною розкладкою та рідкокристалічних моніторів, які здійснюють менш негативний вплив на стан здоров'я працівників відділу.

Оформлення інтер'єру приміщення є відповідає вимогам з ергономіки та стимулює працівників до підвищення працездатності та зниження втоми. Стеля білого кольору створює оптичний ефект збільшення висоти приміщення, підлога пофарбована коричневим кольором, а стіни – у жовтий. Перевагами даного кольору є створення відчуття теплоти, здатність привертати увагу без додаткової втоми.

Висота столу складає 72,5 см, до того ж його можна регулювати відповідно до власних потреб. Стіл має достатній внутрішній об'єм, завдяки ширині у 73 см та висоті простору під столом – у 64 см, є достатньо важким для забезпечення стійкості. Крісла забезпечують фізіологічно раціональну позу, мають підлокітники, здатні обертатися та регулятор висоти, кута нахилу спинки й відстані спинки від краю сидіння.

В кабінеті створено оптимальні умови праці відносно температури, вологості приміщення та вентиляції.

Наприкінці аналізу небезпечних факторів праці побудуємо підсумкову таблицю 8.1.

Таблиця 8.1 – Підсумкова таблиця значень параметрів небезпечних факторів праці

Найменування параметра	Значення параметра		Нормативний документ
	Фактичне	Нормоване	
1	2	3	4
Освітленість штучна, лк	300	300	ДБН.В 2.5-28:2018 [2]
Значення КПО,%	1,0	1,1	ДБН.В 2.5-28:2018 [2]
Повітрообмін,м /год			
взимку	76	80	ДСН 3.3.6.042-99[3]
влітку	36	80	ДСН 3.3.6.042-99 [3]
Температура повітря. °С			
взимку	22	21-25	ДСанПіН 3.3.2-007-98
влітку	24	27-28	ДСанПіН 3.3.2-007-98
Відносна вологість,%			
взимку	60	<75	ДСанПіН 3.3.2-007-98
влітку	55	<60	ДСанПіН 3.3.2-007-98
Швидкість переміщення повітря, м/с			
взимку	0,16	<0,2	ДСанПіН 3.3.2-007-98
влітку	0,10	<0,2	ДСанПіН 3.3.2-007-98

Щодо вимог електробезпеки, то приміщення за небезпекою ураження електричним струмом можна віднести до 1 класу, тобто це приміщення без підвищеної небезпеки (сухе, без пилу, з нормальною температурою повітря, ізольованими підлогами і малим числом заземлених приладів).

Для запобігання поразки електричним струмом в приміщенні відділу використовується ряд організаційно-технічних заходів: розташування проводів

живлення поза зоною пересування людей; допуск до роботи електроприладів тільки тих робітників, що знайомі із технікою безпеки; використання мережних продовжувачів з вбудованими запобіжниками на 0,1 А; при ремонті обладнання персонал попереджується.

Устаткування, що працює в приміщенні живиться від мережі 220В та частотою 50Гц. Споживачами цієї напруги є також джерела штучного освітлення. Вони розташовуються на висоті 3 м, що задовольняє нормі, відповідно до якого джерела освітлення повинні розташовуватися на висоті 2,5 м від підлоги.

Проводка схована. У якості розеток для підключення устаткування застосовуються розетки з заземленим кожухом, захищеного від випадкового доторку до струмоведучих частин. Електроустаткування, що знаходиться в приміщенні відділу відноситься до установок напругою до 1000В.

На робочому місці програміста з всього устаткування металевим є лише корпус системного блоку комп’ютера, але тут використовуються системні блоки, що відповідають стандартові фірми IBM, у яких крім робочої ізоляції передбачений елемент для заземлення і провід з жилою, що заземлює, для приєднання до джерела живлення.

Основні причини ураження людини електричним струмом на робочому місці:

- дотик до металевих неструмоведучих частин (корпусу, периферії комп'ютера), що можуть виявитися під напругою в результаті ушкодження ізоляції:
- нерегламентоване використання електричних приладів:
- відсутність інструктажу співробітників з правил електробезпеки.

На протязі роботи на корпусі комп'ютера накопичується статична електрика. На відстані 5-10 см від екрана напруженість електростатичного поля складає 60-280 кВ/м, тобто в 10 разів перевищує норму 20 кВ м.

Отже за результатами проведеного аналізу можна зробити висновки, що всі показники знаходяться у межах запропонованих значень

8.3 Розробка заходів пожежної безпеки

Ступінь вогнестійкості будинків приймається в залежності від їхнього призначення, категорії по вибухопожежній і пожежній небезпеці, по поверховості, площі поверху в межах пожежного відсіку згідно ДСТУ Б В.1.1-36:2016 Визначення категорій приміщень, будинків, установок за вибухопожежною та пожежною небезпекою [4]

За критеріями по пожежній безпеці будівля відноситься до категорії Д. За особливостями функціонування установи вибухонебезпечних парів та концентратів не помічено. Будівля побудована мармуроподібного вапняка межа стійкості складає 0,5-2,5 години.

Для профілактики пожежі надзвичайно важлива правильна оцінка пожежонебезпеки будинку, визначення небезпечних факторів і обґрунтування способів і засобів пожежопопередження і захисту.

Одне з умов забезпечення пожежобезпеки – ліквідація можливих джерел запалення.

У приміщенні програмістів джерелами запалення можуть бути:

- несправне електроустаткування, несправності в електропроводці, електричних розетках і вимикачах. Для виключення виникнення пожежі з цих причин необхідно вчасно виявляти й усувати несправності, проводити плановий огляд і вчасно усувати всі несправності;
- несправні електроприлади. Необхідні міри для виключення пожежі містять у собі своєчасний ремонт електроприладів, якісне виправлення поломок, не використання несправних електроприладів;
- обігрівання приміщення електронагрівальними приладами з відкритими нагрівальними елементами. Відкриті нагрівальні поверхні можуть спричинити пожежу, тому що в приміщенні знаходяться паперові документи, а папір – легкозаймистий предмет. З метою профілактики пожежі пропоную не використовувати відкриті обігрівальні прилади в приміщенні лабораторії;

- коротке замикання в електропроводці. З метою зменшення імовірності виникнення пожежі внаслідок короткого замикання необхідно, щоб електропроводка була схованою.

- влучення в будинок блискавки. У літній період під час грози можливе влучення блискавки внаслідок чого можливий пожежа. Щоб уникнути цього я рекомендую установити на даху будинку блискавковідвід;

- недотримання мір пожежної безпеки і паління в приміщенні також може спричиниш пожежу. Для усунення загоряння в результаті паління в приміщенні лабораторії пропоную категорично заборонити паління, а дозволити тільки в строго відведеному для цього місці.

За протипожежним режимом визначені: місця куріння, правила проїзду та стоянки транспортних засобів, порядок прибирання пилу й відходів, відключення від мережі електрообладнання у разі пожежі, огляду і закриття приміщень після закінчення операційного дня, проходження посадовими особами навчання та тестування знань з питань пожежної безпеки, проведення з програмістами протипожежних інструктажів, організації експлуатації і обслуговування технічних засобів протипожежного захисту, проведення планово-попереджувальних ремонтів і оглядів інженерного обладнання, дії працівників при виявленні пожежі.

8.4 Розробка заходів по електробезпеці для приміщень з ПЕОМ

Згідно ДБН В.2.5 -28:2018 Державних будівельних норм ДБН В.2.5 -28:2018 [5] при проектуванні систем електропостачання, при монтажі силового електроустаткування і електричного освітлення і в будівлях і приміщеннях для ЕОМ необхідно дотримуватися вимог нормативно-технічної документації.

ПЕОМ є однофазним споживачем електроенергії, що живиться змінним струмом напругою 220В і частотою 50Гц, від мережі із заземленою нейтраллю. За засобами захисту людини від поразки електричним струмом ЕОМ повинне відповідати першому класу захисту. Захист від випадкового дотику до

струмоведучих частин забезпечують конструктивні, схемно-конструктивні і експлуатаційні заходи захисту. Комплекс необхідних заходів по електробезпеці визначається, виходячи з видів електроустановки, її номінальної напруги, умов середовища, типу приміщення і доступності електроустаткуванні.

В приміщенні розміщено декілька комп'ютерів, то кабель прокладають в металевих трубах і гнучких металевих рукавах з відведеннями. Якщо ЕОМ розміщені в центрі приміщення, електромережа прокладається в каналах або під знімною підлогою в металевих трубах і гнучких металевих рукавах.

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

Приміщення з ПК повинні мати природне і штучне освітлення, яке відповідало б вимогам ДБН В.2.5-28-2006 «Природне і штучне освітлення» [1], ДСАНПІН 3.3.2.007-98 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2]. Приміщення для роботи із ПЕОМ повинні мати природне й штучне освітлення. Віконні прорізи повинні бути орієнтовані на північ або на північний схід, забезпечувати коефіцієнт природної освітленості (К.П.О.) не менш 1,5% і мати жалюзі або штори. Віконні прорізи повинні мати регульовані пристрої для відкривання, а також жалюзі, завіски, зовнішні козирки тощо. Приміщення із ПЕОМ повинні бути обладнані системою загального рівномірного освітлення. У виробничих і адміністративно-суспільних 130 приміщеннях, де переважно ведеться робота з документами, допускається комбінована система штучного освітлення. Штучне освітлення має здійснюватися системою загального рівномірного освітлення, яка включає суцільні або такі, що перериваються лінії світильників, розташованих збоку робочих місць (переважно ліворуч), паралельно лінії зору користувачів ПК. Світильники повинні мати розсіювачі світла. У світильниках місцевого освітлення можна використовувати лампи накалювання. При розміщенні ПК по периметру

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

Рисунок 8.1 – Вимоги до системи освітлення робочого місця користувача ПК

Освітленість на робочому столі користувача в зоні розташування документів має бути в межах 300-500 лк. Якщо цей рівень освітленості неможливо забезпечити системою загального освітлення то допускається застосування світильників місцевого освітлення, але при цьому не повинно бути відблисків на поверхні екрану (яскравість відблисків не повинна перевищувати 40 кд/м^2) та перевищення його освітленості більше ніж 300 лк. Яскравість світильників загального освітлення, а також яскравість стелі при застосуванні системи відбитого освітлення не повинна перевищувати 200 кд/м^2. Величина коефіцієнта пульсації освітленості не повинна перевищувати 5%. Що стосується розподілу яскравості в полі зору працюючих за дисплеями ПК, то відношення значень

яскравості робочих поверхонь не повинно перевищувати 3:1

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

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

$$F=ESKZ/n,$$

де: *F* – світловий потік, що розраховується, Лм;

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

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

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

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

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці [7]; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін **($\rho_{стін.}$)** і стелі **($\rho_{стелі}$),** значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

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

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

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

A – ширина приміщення, A = 6 м;

B – довжина приміщення, B = *6,8* м.

Підставимо всі значення у формулу та визначемо індекса приміщення:

$$i=1,1.$$

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

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

Число ламп визначається по формулі:

$$N=F/F_л$$

де: F – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначемо індекса приміщення: N= *43904 / 7200=6,09 шт.*

Приймаємо необхідну кількість *світлодіодних світильників* 7 шт.

Висновки до розділу

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

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

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи підвищення надійності зберігання даних у хмарі.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів підвищення надійності зберігання даних у хмарі.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем підвищення надійності зберігання даних у хмарі.

– Досліджена система підвищення надійності зберігання даних у хмарі.

– На основі отриманих результатів досліджень створена програмна реалізація системи підвищення надійності зберігання даних у хмарі.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання підвищення надійності зберігання даних у хмарі.

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

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

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

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ММВ.

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

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 66437 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,17 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Омелян Ю.В. Дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2023.

2. Stan Stringfellow , Miroslav Klivansky , Michael Barto , Michael Barton – Backup and Restore Practices for the Enterprise. – Prentice Hall PTR., ISBN: 013089401X.

3. Allan N. Packer – Configuring and Tuning Databases on the Solaris Platform. – Sun Microsystems Press., ISBN: 0130834173.

4. Configuring and Tuning Oracle Storage with VERITAS Database Edition for Oracle. Best Practices for Optimizing Performance and Availability for Oracle Databases on Solaris.

5. Alyssa Miller. Cybersecurity Career Guide. Manning Publications. 2022. 368 p.

6. Awais Rashid, Howard Chivers, George Danezis, Emil Lupu, Andrew Martin. CyBOK The Cyber Security Body of Knowledge. The National Cyber Security Centre. 2019. 854 p.

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

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

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

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

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

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

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

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

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

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

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

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

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

20. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020*

IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020, Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

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

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

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

24. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

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

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

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

28. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable*

Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

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

30. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

31. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

32. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

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

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

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

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

37. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019.

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

39. Smirnov, O., Kuznetsov, A., Kiian, A., Gorbenko, Y., Cherep, O., Bexhter L. «Code-based Pseudorandom Generator for the Post-Quantum Period», *2019 IEEE International Conference on Advanced Trends in Information Theory (IEEE ATIT 2019).* 18.12.19-20.12.19 Kyiv Ukraine. P. 204 – 209.

40. Smirnov, O., Kuznetsov, A., Nariezhnii, O., Stelnyk, S., Kokhanovska, T., Kuznetsova T., «Side Channel Attack on a Quantum Random Number Generator», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18 - 21 September 2019. P.713-718.

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

42. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing*

Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P.701-706.

43. Smirnov, O., Hu, Z., Vasiliu, Y., Sydorenko, V., Polishchuk, Y., «Abstract Model of Eavesdropper and Overview on Attacks in Quantum Cryptography Systems», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.399-405.

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

45. Smirnov, O., Kuznetsov, A., Kiian, A., Babenko, B., Zhosan, H., Prokopovych-Tkachenko, D., «Soft Decoding Method for Turbo-Productive Codes», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT 2019*, Lviv, Ukraine, 2-6 July, 2019, P. 129-134.

46. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

47. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

48. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

49. Smirnov, O., Kuznetsov, A., Kiian, A., Kuznetsova, K., Ivko, T., Prokopovych-Tkachenko, D., «Soft Decoding Based on Ordered Subsets of

Verification Equations of Turbo-Productive Codes», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 873-884.

50. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

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

52. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

53. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

54. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи підвищення надійності зберігання даних у хмарі.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 33-13 від 04.08.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи підвищення надійності зберігання даних у хмарі.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

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

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи підвищення надійності зберігання даних у хмарі;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і 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 Visual C#.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Економічні вимоги

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

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2023 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинна бути розглянута розробка заходів пожежної безпеки.

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 – 120 аркушів.

10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2023 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 12.12.2023 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 підвищення надійності зберігання даних у хмарі

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 56

Літера: РП

Кропивницький – 2023 року

Файл Code_Rid_Solomon_Encoder.cs - кодер коду Ріда-Соломона для підвищення надійності зберігання даних на сервері

```
using System;
using System.Threading;

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

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

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

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

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

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

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

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

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

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

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

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

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

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

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

                this.configIsOK = true;
```

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

            return this.configIsOK;
        }

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

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

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

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

                ecc[i] = mulSum;
            }

            return true;
        }

        #endregion Public Operations

        #region Private Operations

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

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

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

                        return;
                    }

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

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

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

                        return;
                    }
                }

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

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

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

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

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

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

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
```

```
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;

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

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

                        return;
                    }
                }

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

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

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

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

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

        #endregion Private Operations
    }
}
```

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

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

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

        #region Windows Form Designer generated code

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

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

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

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

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

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

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

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

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

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

        }

        #endregion

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

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

Файл FileAnalyzer.cs - контроль цілісності даних

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

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

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

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

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

        #endregion Delegates

        #region Public Properties & Data

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

                } else
                {
                    return false;
                }
            }
        }

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

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

        } else
        {
            return false;
        }
    }
}

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

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

        } else
        {
            return false;
        }
    }
}

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

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

        } else
        {
            return null;
        }
    }
}

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

/// <summary>
```

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

                } else
                {
                    return false;
                }
            }
        }

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

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

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

                            break;
                        }

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

                            break;
                        }

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

                            break;
                        }

                        case 3:
                        {
```

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

                            break;
                        }

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

                            break;
                        }
                    }

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

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

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

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

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

        #endregion Public Properties & Data

        #region Data

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

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

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

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

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

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

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

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

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

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

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

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

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

        #endregion Data

        #region Construction & Destruction

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

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

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

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

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

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

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

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

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

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

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

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

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

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

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

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

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

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

                return false;
            }

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

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

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

                return false;
            }

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

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

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

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

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

                // Повертаємо результат обробки
                return this.processedOK;
            }

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

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

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

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

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

            // Повідомляємо, що все нормально
            return true;
        }

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

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

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

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

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

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

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

                // Установлюємо подію завершення обробки
```

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

                return false;
            }

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

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

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

                return false;
            }

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

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

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

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

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

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

                // Повертаємо результат обробки
                return this.processedOK;
            }

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

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

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

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

            // Повідомляємо, що все нормально
            return true;
        }

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

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

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

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

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

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

        #endregion Public Operations

        #region Private Operations

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

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

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

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

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

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

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

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

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

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

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

                            continue;
                        }

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

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

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

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

                            return;
                        }
```

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

                    } // while(true)

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

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

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

                    return;
                }

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

                    } else
                    {
                        break;
                    }
                }

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

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

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

                    return;
                }

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

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

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

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

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

                    return;
                }
            }

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

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

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

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

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

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

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

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

            // Індекс у масиві томів
            int volListIdx = 0;

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

            // Лічильник кількості ушкоджених основних томів
```

```
            int dataVolMissCount = 0;

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

            // Ім'я файлу для обробки
            String fileName;

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

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

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

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

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

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

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

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

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

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

                                    continue;
                                }

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

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

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

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

                                    return;
                                }

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

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

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

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

                            return;
                        }

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

                            } else
                            {
```

```
                                break;
                            }
                        }

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

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

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

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

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

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

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

                        return;
                    }
                }

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

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

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

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

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

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

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

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

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

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

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

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

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

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

                                continue;
                            }

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

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

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

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

                                return;
                            }

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

                        } // while(true)

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

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

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

                        return;
                    }

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

                        } else
                        {
                            break;
                        }
                    }

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

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

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

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

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

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

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

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

                        return;
                    }
                }

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

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

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

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

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

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

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

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

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

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

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

                return;
            }

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

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

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

                return;
            }

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

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

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

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

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

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

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

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

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

        #endregion Private Operations
    }
}
```

Файл MainForm.cs - головне вікно програми

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

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

        #region Windows Form Designer generated code

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

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

```
            this.helpToolStripMenuItem});
            this.menuStrip.LayoutStyle =
System.Windows.Forms.ToolStripLayoutStyle.Flow;
            this.menuStrip.Location = new System.Drawing.Point(0, 0);
            this.menuStrip.Name = "menuStrip";
            this.menuStrip.Size = new System.Drawing.Size(986, 21);
            this.menuStrip.TabIndex = 0;
            this.menuStrip.Text = "menuStrip";
            //
            // fileToolStripMenuItem
            //
            this.fileToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.exitToolStripMenuItem});
            this.fileToolStripMenuItem.Name = "файлToolStripMenuItem";
            this.fileToolStripMenuItem.Size = new System.Drawing.Size(45, 17);
            this.fileToolStripMenuItem.Text = "Файл";
            //
            // instrumentsToolStripMenuItem
            //
            this.instrumentToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.testspeedToolStripMenuItem});
            this.instrumentToolStripMenuItem.Name =
"instrumentsToolStripMenuItem";
            this.instrumentToolStripMenuItem.Size = new System.Drawing.Size(82,
17);
            this.instrumentToolStripMenuItem.Text = "Інструменти";
            //
            // adjustmentToolStripMenuItem
            //
            this.adjustmentToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.encodingfilterToolStripMenuItem,
            this.quickextractionToolStripMenuItem});
            this.adjustmentToolStripMenuItem.Name =
"adjustmentToolStripMenuItem";
            this.adjustmentToolStripMenuItem.Size = new System.Drawing.Size(74,
17);
            this.adjustmentToolStripMenuItem.Text = "Параметри";
            //
            // encodingfilterToolStripMenuItem
            //
            this.encodingfilterToolStripMenuItem.ImageScaling =
System.Windows.Forms.ToolStripItemImageScaling.None;
            this.encodingfilterToolStripMenuItem.Name =
"encodingfilterToolStripMenuItem";
            this.encodingfilterToolStripMenuItem.Size = new
System.Drawing.Size(216, 22);
            this.encodingfilterToolStripMenuItem.Text = "Шифруючий фільтр";
            this.encodingfilterToolStripMenuItem.Click += new
System.EventHandler(this.encodingfilterToolStripMenuItem_Click);
            //
            // helpToolStripMenuItem
            //
            this.helpToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.separatorToolStripMenuItem,
            this.aboutToolStripMenuItem});
            this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
            this.helpToolStripMenuItem.Size = new System.Drawing.Size(60, 17);
            this.helpToolStripMenuItem.Text = "Довідка";
            //
            // separatorToolStripMenuItem
            //
            this.separatorToolStripMenuItem.Name = "separatorToolStripMenuItem";
            this.separatorToolStripMenuItem.Size = new System.Drawing.Size(163,
6);
            //
```

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

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

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

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

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

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

        }

        #endregion

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

Файл Code_Rid_Solomon_Decoder.cs - декодер коду Ріда-Соломона

```
using System;
using System.Threading;

namespace Data_Center
{
    /// <summary>
    /// Клас декодера коду Ріда-Соломона
    /// </summary>
    public class Code_Rid_Solomon_Decoder : Code_Rid_Solomon_Base
    {
        #region Data

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

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

        #endregion Data

        #region Construction & Destruction

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

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

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

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

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

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

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

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

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

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

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

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

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

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

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

                this.configIsOK = true;

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

            return this.configIsOK;
        }

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

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

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

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

                    data[i] = mulSum;
```

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

            return true;
        }

        #endregion Public Operations

        #region Private Operations

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

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

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

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

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

                // Зсув у масиві до елементів k-ой рядка
```

```
                int k_n = k * this.n;

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

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

                // Якщо розв'язний елемент дорівнює нулю - матриця не має
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)
                     &&
                        (OnUpdateCode_Rid_Solomon_MatrixFormingProgress != null)
                   )
                {
                    //...виводимо дані
                    OnUpdateCode_Rid_Solomon_MatrixFormingProgress((((double)(k
+ 1) / (double)this.n) * percOfSecondStage) + percOfFirstStage);
                }

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

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

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

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

                    return false;
                }
            }

            return true;
        }

        /// <summary>
        /// Обчислення логарифмів значень інвертованої матриці
        /// <summary>
        private void LogFCalc()
        {
            // Працюємо з рядками...
            for (int i = 0; i < this.n; i++)
            {
                // Зсув у масиві до елементів і-ой рядка
                int i_n = i * this.n;

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

                    this.FLog[idx] = this.eGF16.Log(this.FLog[idx]);
                }
            }
        }

        /// <summary>
```

```
        /// Заповнення матриці "FLog" (матриці декодера) даними
        /// </summary>
        protected override void FillFLog()
        {
            // Якщо довжина вектора наявних томів менше кількості,
            // необхідного для відновлення...
            if (this.volList.Length < this.n)
            {
                //...указуємо на помилку конфігурації
                this.configIsOK = false;

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

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

                return;
            }

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

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

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

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

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

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

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

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

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

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

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

                    return;
```

```
                }
            }

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

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

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

                    return;
                }
            }

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

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

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

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

                        return;
                    }

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

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

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

                        return;
                    }
                }

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

```
            // Для кожного загубленого основного тому шукаємо том для
відновлення
            for (int i = 0, j = 0; i < dataVolMissCount; i++)
            {
                // Рухаємося за списком томів доти, поки не знайдемо том для
                // відновлення для затикання "дірки" (основні томи мають номера
                // менше this.n (при нумерації з нуля!))
                while (this.volList[j] < this.n)
                {
                    j++;
                }

                // Зберігаємо номер тому для заміни загубленого основного тому
                eccVolToFix[i] = this.volList[j];

                j++; // j++ дозволяє перейти до наступного пошуку
            }

            // Працюємо по рядках матриці (в ідеалі, всі рядки повинні
заповнюватися
            // рядками з одиницею на головній діагоналі, що відповідає
відсутності
            // ушкоджень, але allVolCount укаже, якими є справи з наявністю
томів)
            for (int i = 0, e = 0; i < this.n; i++)
            {
                // Індекс рядка з дисперсної матриці, що буде поміщена в матрицю
кодування
                int DRowIdx;

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

                // Якщо основний том відсутній, формуємо рядок матриці
Вандермонда
                if (allVolCount[i] == 0)
                {
                    // Обчислюємо номер рядка матриці Вандермонда, яку потрібно
вставити
                    // на місце даного рядка формованої матриці "FLog"
                    DRowIdx = eccVolToFix[e++];

                    // Указуємо, що даний рядок матриці "FLog" не тривіальна
                    this.FLogRowIsTrivial[i] = false;

                } else
                {
                    // Формуємо в матриці "FLog" нульовий рядок з одиницею на
головній діагоналі
                    // (відповідає наявному основному той)
                    DRowIdx = i;

                    // Указуємо, що даний рядок матриці "FLog" тривіальна
                    this.FLogRowIsTrivial[i] = true;
                }

                // Залежно від типу декодера беремо дані з відповідного масиву
                // (у ньому втримуються як рядки матриці Вандермонда, так і
"тривіальні" рядки,
                //  утримуючі нулі і єдиний елемент "1" на головній діагоналі)
                if (this.eCode_Rid_Solomon_Type ==
(int)Code_Rid_Solomon_Type.Dispersal)
                {
                    int bs = DRowIdx * this.n;

                    // Формування рядка в матриці кодування
                    // ("тривіальні" рядки вже втримуються в матриці "D", вони
вийшли
```

```
                    // "автоматично" на попередньому етапі обробки
MakeDispersal())
                    for (int j = 0; j < this.n; j++)
                    {
                        this.FLog[i_n + j] = this.D[bs + j];
                    }

                } else
                {
                    // Якщо це потрібно - формуємо "тривіальну" рядок...
                    if (this.FLogRowIsTrivial[i])
                    {
                        for (int j = 0; j < this.n; j++)
                        {
                            this.FLog[i_n + j] = 0;
                        }

                        this.FLog[i_n + i] = 1;

                    } else
                    {
                        int bs = (DRowIdx - this.n) * this.n;

                        //...а, інакше, беремо рядок матриці Вандермонда
                        for (int j = 0; j < this.n; j++)
                        {
                            this.FLog[i_n + j] = this.A[bs + j];
                        }
                    }
                }

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

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    //...указуємо на помилку конфігурації
                    this.configIsOK = false;

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

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

                    return;
                }
            }

            // Знаходимо зворотну матрицю для "FLog"
            if (!FInv())
            {
                // Указуємо, що кодер зконфігуровано некоректно
                this.configIsOK = false;

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

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

                return;
            }

            // Обчислюємо логарифми елементів інвертованої матриці
            LogFCalc();
```

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

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

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

        #endregion Private Operations

        #region Public Properties

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

                } else
                {
                    return null;
                }
            }
        }

        #endregion Public Properties
    }
}
```

Файл About.cs - вікно довідки про програму

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

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

        #region Windows Form Designer generated code

        /// <summary>
        /// Необхідний метод для підтримки розробника - не модифікується
        /// зміст цього метода використовується редактором коду.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.developersListBoxdevelopersListBox = new
System.Windows.Forms.ListBox();
            this.Code_Rid_Solomon_IconTimer = new
System.Windows.Forms.Timer(this.components);
            this.okButton_Windows_8 = new PinkieControls.Button_Windows_8();
            this.SuspendLayout();
            //
            // developersListBoxdevelopersListBox
            //
            this.developersListBoxdevelopersListBox.BackColor =
System.Drawing.SystemColors.Control;
            this.developersListBoxdevelopersListBox.BorderStyle =
System.Windows.Forms.BorderStyle.None;
            this.developersListBoxdevelopersListBox.FormattingEnabled = true;
            this.developersListBoxdevelopersListBox.Items.AddRange(new object[]
{
            "МАГІСТЕРСЬКА РОБОТА",
            "",
            "На тему:",
            "",
            " Дослідження та програмна реалізація системи підвищення надійності
зберігання даних у хмарі ",
            "",
            "",
            "Керівник: Якименко Н.М.",
            "",
            "Розробив: студент Омелян Юрій Володимирович",
            "                  гр. КН-22М-2",
            "",
            "м. Кропивницький 2023"});
            this.developersListBoxdevelopersListBox.Location = new
System.Drawing.Point(11, 6);
            this.developersListBoxdevelopersListBox.Name =
"developersListBoxdevelopersListBox";
```

```
            this.developersListBoxdevelopersListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.developersListBoxdevelopersListBox.Size = new
System.Drawing.Size(330, 182);
            this.developersListBoxdevelopersListBox.TabIndex = 0;
            this.developersListBoxdevelopersListBox.TabStop = false;
            this.developersListBoxdevelopersListBox.SelectedIndexChanged += new
System.EventHandler(this.developersListBoxdevelopersListBox_SelectedIndexChanged
);
            //
            // Code_Rid_Solomon_IconTimer
            //
            this.Code_Rid_Solomon_IconTimer.Interval = 40;
            this.Code_Rid_Solomon_IconTimer.Tick += new
System.EventHandler(this.Code_Rid_Solomon_IconTimer_Tick);
            //
            // okButton_Windows_8
            //
            this.okButton_Windows_8.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.okButton_Windows_8.DefaultScheme = true;
            this.okButton_Windows_8.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.okButton_Windows_8.Hint = "";
            this.okButton_Windows_8.Location = new System.Drawing.Point(266,
177);
            this.okButton_Windows_8.Name = "okButton_Windows_8";
            this.okButton_Windows_8.Scheme =
PinkieControls.Button_Windows_8.Schemes.Blue;
            this.okButton_Windows_8.Size = new System.Drawing.Size(75, 23);
            this.okButton_Windows_8.TabIndex = 0;
            this.okButton_Windows_8.Text = "OK";
            this.okButton_Windows_8.Click += new
System.EventHandler(this.okButton_Windows_8_Click);
            //
            // AboutForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(350, 212);
            this.Controls.Add(this.okButton_Windows_8);
            this.Controls.Add(this.developersListBoxdevelopersListBox);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "AboutForm";
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "Про програму...";
            this.Load += new System.EventHandler(this.AboutForm_Load);
            this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.AboutForm_FormClosing);
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.ListBox developersListBoxdevelopersListBox;
        private System.Windows.Forms.Timer Code_Rid_Solomon_IconTimer;
        private PinkieControls.Button_Windows_8 okButton_Windows_8;

    }
}
```