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

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи оцінки продуктивності систем зберігання даних”

Виконав здобувач вищої освіти
IV курсу, групи КІ-22-МБ
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Пацан І.В.
«____» _____ 2025 р.

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Якименко Н.М.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

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

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

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

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

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

Пацану Ігорю Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи оцінки продуктивності систем зберігання даних*

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 № 48-02 від 17.01.2025 року

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

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи оцінки продуктивності систем зберігання даних*

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

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

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

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

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

КАЛЕНДАРНИЙ ПЛАН

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

Дата видачі завдання
« 17 » січня 2025 р.

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

Завдання прийнято до виконання
« 17 » січня 2025 р.

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

АНОТАЦІЯ

Пацан І.В. Програмне забезпечення системи оцінки продуктивності систем зберігання даних. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи оцінки продуктивності систем зберігання даних.

Метою розробки є програмне забезпечення системи оцінки продуктивності систем зберігання даних.

Результат роботи – програмна реалізація системи оцінки продуктивності систем зберігання даних.

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

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

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

Програму розроблено в середовищі Visual C#.

Ключові слова: комп'ютерна інженерія, продуктивність систем зберігання даних

ABSTRACT

Patsan I.V. Software for the performance assessment system of data storage systems. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the performance assessment system of data storage systems.

The purpose of the development is the software for the performance assessment system of data storage systems.

The result of the work is the software implementation of the performance assessment system of data storage systems.

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 the software are provided.

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

The program was developed in the Visual C# environment.

Keywords: computer engineering, performance of data storage systems

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АВІ	–	адміністратора віртуальної інфраструктури
АІБ	–	адміністратор інформаційної безпеки
ВІ	–	віртуальна інфраструктура
ДЦ	–	дата-центр
ЗЗІ	–	засоби захисту інформації
ПД	–	персональні дані
ПЗ	–	програмне забезпечення
СЗД	–	системи зберігання даних
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. Щоб правильно вибрати нову систем зберігання даних СЗД або спланувати модернізацію існуючої, необхідно з'ясувати вимоги, пропоновані IT-системою до зберігання даних. Основні з них – ємність і продуктивність. І якщо питання «Скільки терабайтів корисної ємності вам потрібно?», як правило, не викликає проблем, то прохання вказати необхідну продуктивність можуть багатьох загнати в глухий кут.

Як довідатися продуктивність систем зберігання даних (СЗД)? Існують два підходи для її оцінки – технічний і користувальницький. У першому випадку продуктивність описується рядом технічних параметрів, пов'язаних з роботою СЗД. Такий підхід використовується в основному IT-фахівцями. У другому випадку продуктивність оцінюється на підставі суб'єктивних думок користувачів щодо того, наскільки швидко працює IT-система. Очевидно, що для реальної оцінки продуктивності СЗД цей підхід не годиться, але про нього не треба забувати, оскільки користувачі інформаційних систем бачать продуктивність будь-якого компонента IT-системи крізь призму своїх моніторів.

Отже, з погляду IT-фахівця, продуктивність СЗД – це в першу чергу кількість операцій введення-виводу в секунду (IOPS) і обсяг переданих мегабайтів у секунду (Мбайт/с), які система зберігання здатна забезпечити при читанні й записі даних. Продуктивність СЗД в IOPS використовується для оцінки навантаження транзакційних застосунків: баз даних Online Transaction Processing (OLTP), файлових сховищ, поштових систем і іншого. Інший технічний параметр, тісно пов'язаний із транзакційним навантаженням, – час відгуку при операціях введення-виводу (response time). Іншими словами, цей час, витрачений СЗД на обробку однієї операції введення-виводу й передачу її результатів хосту.

Час відгуку й раніше використовувалося поряд з кількістю IOPS для детального планування конфігурації СЗД. Але широку популярність цей

параметр придбав після появи СЗД, цілком побудованих на базі флеш-накопичувачів. Основна особливість цих систем – здатність обробляти введення-вивід застосунків згодом відгуку менше однієї мілісекунди. Для ряду застосунків, зокрема баз даних OLTP, мінімально можливий час відгуку так само важливо, як і IOPS.

Для оцінки продуктивності застосунків, у яких профіль навантаження на СЗД являє собою послідовний доступ до даних, прийнято використовувати обсяг переданих даних, виражений у мегабайтах у секунду (Мбайт/с). Приклад таких застосунків – бази даних у конфігурації «сховище даних» (Data Warehouse, DWH), застосунки для обробки відеоконтента й резервного копіювання.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи оцінки продуктивності систем зберігання даних.

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

– Огляд існуючих систем оцінки продуктивності систем зберігання даних.

– Дослідження системи оцінки продуктивності систем зберігання даних.

– Програмна реалізація системи оцінки продуктивності систем зберігання даних.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі оцінки продуктивності систем зберігання даних.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи оцінки продуктивності систем зберігання даних, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для оцінки продуктивності систем зберігання даних. У сучасних інформаційних інфраструктурах типу "хмар" або великих обчислювальних кластерів значний вплив на загальну продуктивність і якість роботи робить підсистема зберігання даних. На відміну від обчислювальної потужності й обсягів пам'яті, нарощуваних шляхом додавання серверів в інфраструктуру – сховище значно складніше піддається масштабуванню, що змушує особливо ретельно підходити до підбору його конфігурації.

Масування застосування технологій віртуалізації й широке поширення SSD-Носіїв створили одночасно й нові потреби й нові можливості в області побудови систем зберігання даних. Як наслідок – останнім часом з'явився ряд нових способів організації системи зберігання даних, у яких активно застосовуються нетривіальні підходи, такі як багаторівневе кешування, гетерогенні носії, віртуалізація системи зберігання. Їхнє поводження під різним навантаженням не завжди прогнозована й оцінка ефективності застосування того або іншого рішення вимагає розширеного тестування.

Архітектура систем зберігання

Традиційні системи зберігання

DAS (direct attached storage) – сховище із прямим підключенням. Звичайно представляє із себе RAID-контролер до якого підключаються кошики з дисками. Кошики можуть бути як убудованими безпосередньо в сервер, так і в додаткове шасі зі своїм джерелом живлення.

SAN (storage area network) – мережа зберігання даних (СЗД). Це один або кілька дискових масивів із власними RAID-контролерами (найчастіше дубльованими) і додатковими дисковими полками, об'єднаних за допомогою

мережного встаткування. Найпоширеніші спеціалізовані оптоволоконі мережі із протоколом FC і мережі на базі Ethernet с протоколом iSCSI. У невеликих СЗД можливо використання протоколу SAS. Доступ до СЗД із боку серверів здійснюється за допомогою спеціальних адаптерів (HBA – host bus adapter). HBA не є RAID-контролером. У випадку використання протоколу iSCSI роль HBA може виконувати звичайна мережна карта.

NAS (network attached storage) – сховище, що підключається по мережі. Фактично – це файл-сервер, інтегрований з DAS або SAN.

Системи зберігання, засновані на трансляції адрес

NetApp ZFS

У прагненні уникнути недоліків, властивих традиційним RAID-алгоритмам, у компанії NetApp винайшли WAFL (Write Anywhere File Layout) – систему розміщення даних по дисках, що поєднує функції файлової системи, менеджера томів і механізму захисту даних за допомогою контрольних сум. Аналогічну технологію запропонувала фірма Sun пізніше поглинена Oracle у своїй файловій системі ZFS.

Віртуалізатори (IBM SAN Volume Controller, quadstor) – це програмні (або програмно-апаратні) надбудови над існуючими системами зберігання. Основна можливість – подання безлічі пристроїв зберігання як єдиного віртуального сховища, що можливо більш гнучко розподіляти між споживачами. Оскільки в основі роботи віртуалізаторів у тому або іншому ступені лежить механізм трансляції адрес – ця технологія дозволяє виконувати дедуплікацію даних.

З масованим застосуванням технологій віртуалізації для десктопної інформаструктури (VDI – virtual desktop infrastructure) до систем зберігання даних з'явилася нова вимога – ефективно підтримувати більші кількості слабковідрізняючихся наборів даних (образи віртуальних машин в основному є численними копіями операційної системи).

SSD

Кластерні системи

Для високопродуктивний обчислювальних систем і систем, орієнтованих на обробку транзакцій більше важливою характеристикою продуктивності є латентність, тобто час очікування реакції на запит. Попередні способи організації СЗД використовують добре масштабовані, але відносно повільні інтерфейси між застосунком, що споживає дані й властиво дисками. Для спеціалізованих кластерних СЗД характерне використання дискових масивів на основі SSD, пряме підключення масивів до обчислювальних вузлів за допомогою стандартних HBA або HBA, що забезпечують зв'язність самого кластера (наприклад Infiniband) і використання спеціалізованих файлових систем для об'єднання ємності декількох масивів у єдиний простір.

1.2 Область застосування

Областю застосування є систем зберігання даних.

В області тестування сховищ даних галузевим, всім визнаним стандартом є бенчмарк SPC (Storage Performance Council), зокрема SPC-1, що оцінює продуктивність системи в IOPS – кількості операцій введення-виводу в секунду. Результати незалежних тестів устаткування різних виробників регулярно публікуються на сайті SPC :

http://www.storageperformance.org/results/benchmark_results_spc1.

На жаль інструментарій для проведення тестів SPC доступний тільки членам даної організації, що робить практично неможливим повторення тесту або оцінку впливу на продуктивність системи конфігурації СЗД або інших елементів інфраструктури.

Оскільки продуктивність сильно залежить як від конкретної конфігурації, так і від характеру навантаження – існує потреба в тестах, які може запустити кінцевий користувач. Одним з таких засобів тестування продуктивності є утиліта IOMeter, спочатку розроблена Intel, а потім передана open source-співтовариству. Докладно робота з IOMeter описана, наприклад, на http://itband.ru/2010/01/iometer.

IOMeter є синтетичним тестом, що дозволяє гнучко варіювати профіль створюваного навантаження: можна міняти розмір блоків введення-виводу, характер доступу (випадкова або послідовний), кількість паралельних операцій, а також комбінувати різні профілі, імітуючи навантаження, характерну для реальних застосунків. Можливе проведення розподіленого тестування, коли навантаження генерується відразу декількома машинами, що дозволяє наблизити умови тесту до реального інформаційного простору.

У поставку IOMeter входять типові вирождені тести, наприклад:

– випадкове читання блоків по 4 Кб;

– послідовне читання блоків по 4 Кб;

– випадковий запис блоків по 4 Кб;

– послідовний запис блоків по 4 Кб.

Також доступні тестові паттерни Intel і StorageReview, призначені для створення навантаження, характерної для деяких реальних застосувань.

– NFS-сервер;

– сервер БД;

– web-сервер.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи оцінки продуктивності систем зберігання даних, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

EMC Avamar

EMC Avamar: швидке, ефективне резервне копіювання й відновлення за допомогою комплексного програмного й апаратного рішення. Рішення Avamar з інтегрованою технологією дедуплікації сегментів даних змінної довжини підтримує швидке щоденне повне резервне копіювання для віртуальних середовищ, віддалених офісів, корпоративних застосунків, серверів NAS, настільних комп'ютерів і ноутбуків.

Прискорене резервне копіювання й відновлення

В Avamar застосовується дедуплікація сегментів даних змінної довжини, що дозволяє значно зменшити час резервного копіювання, тому що зберігаються тільки унікальні зміни, внесені за день. При цьому створюються щоденні повні резервні копії даних для негайного відновлення в один етап.

Оптимізована смуга пропущення

При резервному копіюванні з дедуплікацією передаються тільки змінені блоки, що приводить до зменшення обсягу мережного трафіку. Використовуйте існуючу смугу пропущення локальної й глобальної мереж для резервного копіювання й відновлення даних усього підприємства, а також віддалених офісів і філій.

Відновлення за один крок

Кожна резервна копія, створена Avamar, є повної, що дозволяє дуже легко знайти й вибрати потрібну резервну копію, щоб запустити відновлення в один етап.

Висока надійність

Рішення Avamar Data Store оснащене резервованими системами живлення й мережних інтерфейсів, захистом RAID, а також патентованою технологією RAIN для забезпечення безперебійного доступу до даних. Щоденні перевірки систем зберігання даних гарантують можливість відновлення в будь-який час.

Гнучке розгортання

Avamar Data Store масштабується до 124 Тбайт дедуплікованої ємності. Для подальшого підвищення продуктивності й масштабованості Avamar може розгортатися як інтегроване рішення разом із системами EMC Data Domain.

EMC Avamar забезпечує швидке й ефективне резервне копіювання й відновлення завдяки зменшенню обсягу даних резервного копіювання на стороні клієнта ще до того, як вони будуть передані по мережі й збережені. Дедуплікація сегментів даних змінної довжини, виконувана Avamar, значно зменшує обсяг мережного трафіку, тому що по локальних і глобальних мережах передаються тільки унікальні блоки даних у стислому й зашифрованому виді. Попередньо збережені блоки більше не піддаються резервному копіюванню.

Це забезпечує величезну економію ресурсів смуги пропущення при резервному копіюванні, значне скорочення необхідних обсягів дискових ресурсів зберігання й, що найважливіше, дуже швидке виконання резервного копіювання: найчастіше в десять разів швидше.

Резервні копії Avamar можна швидко відновити всього за один крок: для досягнення необхідної крапки відновлення не потрібно відновлювати спочатку повну, а потім наступні інкрементні резервні копії. Для підвищення безпеки можливо також шифрування даних резервного копіювання на Avamar Data Store.

Варіанти розгортання

Avamar Data Store – найпростіший і швидкий варіант розгортання цього рішення. EMC Avamar Data Store поєднує сертифіковане EMC спеціалізований пристрій для резервного копіювання й ПЗ для резервного копіювання й відновлення Avamar з функцією дедуплікації в повністю інтегрованому й

масштабованому готовому рішенні. Воно дозволяє уникнути труднощів, які виникають при роботі з багатьма постачальниками встаткування, програмного забезпечення й послуг підтримки. Готове до негайної експлуатації рішення Avamar Data Store значно зменшує час налаштування на місці. Щоб його придбати, розгорнути й забезпечити обслуговування, досить зв'язатися з однією контактною особою.

Avamar Business Edition – спеціалізований пристрій резервного копіювання EMC Avamar Business Edition пропонує компактне, готове до використання й доступне рішення для резервного копіювання з функцією дедуплікації. Дане рішення розроблене для компаній середнього розміру й забезпечує спрощене керування, що робить його ідеальним для організацій з обмеженими IT-ресурсами. Убудована відказостійкість системи зберігання й застосункова можливість реплікації гарантують доступність і забезпечують аварійне відновлення.

Avamar Virtual Edition – ПЗ для резервного копіювання Avamar з функціями дедуплікації й віртуальним пристроєм, розгорнутий в vSphere або Hyper-V і Azure.

Інтеграція із системами зберігання Data Domain з функцією дедуплікації – використовуйте продуктивність і масштабованість рішень Data Domain для всіх робочих навантажень резервного копіювання.

Модулі застосунків

EMC Avamar захищає критично важливі для бізнесу застосунка без переривань або простоїв. Це рішення забезпечує гнучке відновлення даних, необхідне для швидкого поновлення бізнес-операцій. Використання функції виключення дублікатів даних забезпечує захист даних ключових бізнес-застосунків за менший час, максимально ефективне використання мережі під час резервного копіювання й мінімальне споживання ємності сховища резервних копій.

Таблиця 2.1 – Сценарії використання

Віртуалізировані середовища	Оптимізуйте резервне копіювання й відновлення віртуальних машин як на рівні гостьових ОС, так і на рівні образів. Використовуйте унікальні можливості й тісну інтеграцію з VMware vCenter, VMware vCloud Director, програмним інтерфейсом VMware vStorage API for Data Protection, vSphere Web Client і Microsoft Hyper-V.
Резервне копіювання мережної системи зберігання даних	Скоротите час резервного копіювання за рахунок використання протоколу NDMP (Network Data Management Protocol). Відмовтеся від тривалого створення базових повних резервних копій. Скористайтеся високопродуктивною передачею декількох потоків для забезпечення функцій резервного копіювання й відновлення в системі EMC Isilon.
Резервне копіювання настільних ПК і ноутбуків	Захищайте дані на периферії й зменшите залежність від IT-фахівців завдяки можливості самостійного відновлення , виконуваного кінцевими користувачами.
Резервне копіювання даних віддалених офісів	Централізуйте й оптимізуйте резервне копіювання й відновлення у віддалених офісах за рахунок єдиного інтерфейсу користувача.
Критично важливі для бізнесу застосунка	Забезпечте резервне копіювання й відновлення з урахуванням застосунків для корпоративних застосунків IBM, Microsoft, Oracle і SAP за допомогою високопродуктивної дедуплікації, а також поліпшену видимість і можливості керування для власників застосунків.

Інтеграція рішень Avamar і EMC Data Domain надає можливість направити обрані робочі навантаження резервного копіювання, які обробляє Avamar, у системи Data Domain для підвищення масштабованості й продуктивності.

Для тісної інтеграції з бізнес-застосунками EMC пропонує наступні модулі застосунків Avamar.

EMC Avamar Client for IBM DB2 – комплексне рішення для резервного копіювання й відновлення баз даних IBM DB2. Модуль забезпечує просте й швидке відновлення баз даних на певний момент часу.

EMC Avamar Client for Lotus Notes – забезпечує швидке резервне копіювання з дедуплікацією і відновлення для Lotus Notes, включаючи підтримку Domino Attachment and Object Service (DAOS).

EMC Avamar Client for Microsoft Exchange Server – забезпечує резервне копіювання з виключенням дублікатів даних і відновлення для Microsoft Exchange Server. Підтримує відновлення окремих поштових скриньок, папок або повідомлень із високим рівнем деталізації.

EMC Avamar Client for Microsoft SharePoint – забезпечує резервне копіювання з виключенням дублікатів даних і відновлення для Microsoft SharePoint. Модуль забезпечує деталізоване відновлення об'єктів SharePoint, таких як веб-застосунка, веб-сайти й календарі.

EMC Avamar Client for Microsoft SQL Server – забезпечує швидке оперативне резервне копіювання для баз даних Microsoft SQL Server з повним аварійним відновленням і можливістю деталізованого відновлення.

EMC Avamar Client for Oracle – забезпечує резервне копіювання й відновлення з виключенням дублікатів даних для середовищ Oracle. Підтримує резервне копіювання Oracle Recovery Manager (RMAN) і забезпечує комплексний захист Oracle Real Application Cluster (RAC).

EMC Avamar Client for SAP – забезпечує повністю функціональне резервне копіювання з дедуплікацією і відновлення даних SAP, включаючи підтримку активних і пасивних кластерів і деталізоване відновлення.

EMC Avamar Client for Sybase – забезпечує повністю функціональне резервне копіювання з дедуплікацією і відновлення для Sybase, включаючи підтримку кластерів «активний/пасивний» і деталізоване відновлення.

Резервне копіювання баз даних служб Microsoft SQL Server Служби Analysis Services

Служби Analysis Services передбачені резервне копіювання й відновлення бази даних і її об'єктів на певний момент часу. Резервне копіювання й відновлення підходить для переносу баз даних на модернізовані сервери, між серверами або для розгортання бази даних на робочому сервері.Якщо є коштовні дані, але поки немає плану резервного копіювання, то необхідно якомога швидше розробити й реалізувати такий план на випадок наступного відновлення даних.

Команди резервного копіювання й відновлення виконуються в розгорнутій базі даних служб Analysis Services. Для проектів і рішень у середовищі SQL Server Data Tools (SSDT)варто використовувати систему керування версіями, що дозволяє відновлювати певні версії вихідних файлів, а потім створювати план відновлення даних для репозиторія використовуваної системи керування версіями.

Щоб створити повну резервну копію, що включає вихідні дані, необхідно створити резервну копію бази даних, що містить докладні дані. Зокрема, якщо використовується режим зберігання бази даних ROLAP або DirectQuery, докладні дані зберігаються в зовнішньої реляційної базі даних SQL Server окремо від бази даних служб Analysis Services. У противному випадку, якщо всі об'єкти є таблична або багатомірними, резервна копія служб Analysis Services буде містити й метадані, і вихідні дані.

Однією з явних вигід автоматизації резервного копіювання є те, що моментальний знімок даних буде завжди настільки оновленим, наскільки це задано частотою автоматичного резервного копіювання. Автоматичні планувальники гарантують, що резервне копіювання буде зроблено вчасно. Відновлення бази даних теж можна автоматизувати, і воно може бути гарним

способом реплікації даних, але необхідно обов'язково створити резервну копію файлу ключа шифрування в екземплярі, на якому здійснюється реплікація. Функція синхронізації призначена для реплікації баз даних служб Служби Analysis Services , але ставиться тільки до застарілих даних. Всі описувані тут функції можуть бути реалізовані через користувальницький інтерфейс за допомогою команд XML/A або запущені програмним шляхом через об'єкти AMO.

Існує кілька способів резервного копіювання баз даних служб Microsoft SQL Server Служби Analysis Services, і всі вони вимагають прав адміністратора сервера й бази даних. Можна відкрити діалогове вікно **Резервне копіювання** в середовищі Середовище SQL Server Management Studio, установити потрібну конфігурацію параметрів, а потім запустити резервне копіювання із самого вікна. Крім цього, можна створити скрипт, використовуючи налаштування, уже задані у файлі. Потім скрипт можна зберегти й запускати так часто, як потрібно.

Резервне копіювання й синхронізація

Якщо база даних розташована на віддаленому екземплярі служб Служби Analysis Services, можна використовувати функцію синхронізації для резервного копіювання бази даних на локальний екземпляр. У такий спосіб можна переносити складання бази даних із середовища розробки у виробниче середовище. Також можна використовувати звичайні операції резервного копіювання й відновлення на рівні файлів, щоб перемістити складання із середовища розробки у виробниче середовище, але синхронізація надає додаткові можливості. Наприклад, параметри безпеки можуть розрізнятися на комп'ютері для розробки й виробничому комп'ютері. Синхронізація надасть можливість зберегти ці налаштування й синхронізувати всі об'єкти, крім ролей. Крім того, синхронізація звичайно робить додаткові відновлення тих об'єктів, які розрізняються на комп'ютерах джерела й призначення. Такий тип додаткового резервного копіювання не доступний для функції резервного копіювання й відновлення.

Існує безліч способів відновити бази даних служб Microsoft SQL Server Служби Analysis Services, кожний з яких вимагає наявності дозволів адміністратора як на серверному комп'ютері, так і в базі даних служб Служби Analysis Services. Щоб відновити базу даних служб Служби Analysis Services , можна відкрити діалогове вікно **Відновлення бази даних** у середовищі Середовище SQL Server Management Studio, вибрати необхідну конфігурацію параметрів, а потім запустити операцію відновлення в діалоговому вікні. Або можна створити скрипт, використовуючи вже задані у файлі налаштування. Скрипт може бути збережений. Він буде запускатися в міру необхідності. Таким чином, операція відновлення виконується з використанням XML для аналітики (див. опис у розділі нижче).

Відновлення баз даних з використанням XML для аналітики

Команда XMLA Restore являє собою спосіб автоматизації процесу відновлення шляхом запуску операції відновлення, заснованої на ABF-файлі. У команди Restore є ряд властивостей, які можна настроїти так, щоб задати визначення безпеки, указати місце зберігання віддалених секцій, а також параметри переміщення реляційних об'єктів OLAP (ROLAP).

Дискові накопичувачі PowerVault RD1000

Знімні дискові касети Dell™ PowerVault™ RD1000 – це простої у використанні, економічне й надійне рішення для зберігання й резервного копіювання даних за допомогою знімних дискових накопичувачів PowerVault RD1000. Знімні касети компактні й зручні для перенесення й відмінно підходять для позаофісних систем зберігання й захисту даних при збоях. По закінченні резервного копіювання можна просто витягти касету й відкласти її на зберігання або взяти із собою. ПЗ безперервного захисту даних, що включено в комплект поставки кожного пристрою PowerVault RD1000, надає користувачам надійну й просту у використанні комплексну систему захисту даних.

Надійні знімні накопичувачі, готові до роботи

Знімна дискова касета PowerVault RD1000 – це надійне рішення, що забезпечує швидкий захист всієї необхідної важливої інформації й дозволяє завжди мати її під рукою. Знімні дискові носії оснащені міцним твердим

корпусом і убудованими амортизаторами. Вони мають достатню міцність, щоб витримати падіння з висоти приблизно одного метра. Убудовані амортизатори касети PowerVault RD1000 дозволяють:

– запобігти впливу вібрації;

– поліпшити захист даних при транспортуванні;

– забезпечують можливість розміщення PowerVault RD1000 як у горизонтальному, так і у вертикальному положенні.

Широкий спектр варіантів ємності

У продажі є знімні дискові касети ємністю 320 Гбайт, 500 Гбайт, 1 Тбайт і 2 Тбайт. Для забезпечення додаткового захисту й більше ефективного відновлення після збоїв можна забезпечити щоденну або щотижневу ротацію декількох знімних дискових касет між місцем установки встаткування й зовнішнім сховищем на випадок екстреної ситуації.

Сумісність накопичувачів

Кожна знімна дискова касета сумісна з усіма стикувальними модулями PowerVault RD1000 (читання/запис), які можуть бути встановлені в різних місцях, тому інформація завжди буде доступна вам поза залежністю від місця розташування.

Ефективність і можливість використання майбутніх технологій

У порівнянні зі стрічковими накопичувачами початкового рівня, знімні дискові накопичувачі мають мінімальну вартість первісної установки й налаштування. Диски й касети більше надійні й довговічні й, отже, скорочують сукупну вартість володіння. Відповідно до результатів тестування приблизний строк експлуатації знімних дискових касет становить 30 років і більше (англійською мовою). Крім того, для PowerVault RD1000 не потрібні касети, що чистять. Це дійсно економічне рішення, що відмінно підходить для використання як в окремих офісах, так і в підрозділах підприємства.

Швидке резервне копіювання й доступ до даних

Dell PowerVault RD1000 характеризується найвищою швидкістю звертання до даних на диску й забезпечує практично миттєве відновлення файлів.

PowerVault DL4000

Швидке й ефективне відновлення даних

Надійно убезпечте дані в міру росту вашого підприємства й будьте впевнені, що у випадку відмови встаткування або ПЗ ви зможете відновити роботу всього за кілька хвилин, а не днів або годин.

Надійний помічник у бізнесі

Скористайтеся не тільки можливостями резервного копіювання, але й інтегрованими функціями відновлення після збоїв з використанням передових технологій ПЗ AppAssure для Microsoft® Windows® і Linux®.

Простота розгортання й висока масштабованість

Здійснюйте швидке розгортання за допомогою спеціалізованих засобів і ефективно розширюйте інфраструктуру за допомогою масштабованих рішень, що забезпечують постійний захист зростаючого середовища бізнес-аналітики.

Глобальне обслуговування й підтримка

Послуги Dell™ допомагають зменшити складність IT-інфраструктури, скоротити витрати й виключити неефективні операції за рахунок оптимізації IT- і бізнес-рішень клієнта. Група фахівців з обслуговування Dell пропонує унікальні можливості, включаючи рішення для повного перенесення даних, які дозволяють спростити консолідацію декількох файлових систем. Фахівці Dell попередньо вивчають всі наявні потреби клієнта й проектують рішення відповідно до конкретної інфраструктури й бізнес-цілями. При цьому залучаються місцеві фахівці, ефективно використовуються перевірені методи й великі знання в конкретній області, що дозволяє скоротити сукупну вартість володіння.

Система зберігання даних зі знімними дисками Dell PowerVault RD1000

У порівнянні з іншими носіями для резервного копіювання даних очевидні переваги знімного диска великої ємності в комплекті з касетою, такого як Dell PowerVault RD1000. Міцний знімний дисковий носій більше зручний при транспортуванні й прослужить довше, ніж стандартні жорсткі диски USB для зберігання даних. Касети PowerVault RD1000 витримують падіння з висоти

майже одного метра. Дуже міцні, компактні й легені, вони можуть зберігатися поза офісом для забезпечення додаткового захисту й відновлення після збоїв. PowerVault RD1000 дозволяє легко збільшити ємність сховища. Коли одна знімна дискова касета заповнюється, просто вставляється нова касета. Здобувати додаткове встаткування не потрібно. PowerVault RD1000 включає програмне забезпечення резервного копіювання, що забезпечує захист даних, прискорює відновлення й знижує час простою.

PowerVault RD1000 включає захисну дискову касету ємністю 320, 500 Гбайт, 1 або 2 Тбайта. Всі знімні дискові касети RD взаємозамінні й попередньо відформатовані. Архіви даних на дискових касетах PowerVault RD1000 займають набагато менше місця, чим архіви на CD- або DVD-дисках: для запису даних обсягом 1 Тбайт, які вміщаються на одній дисковій касеті, треба було б більше 200 стандартних DVD-дисків.

Швидкість і надійність жорсткого диска

Знімна дискова касета PowerVault™ RD1000 містить жорсткий диск Serial ATA (SATA) – ту ж технологію, що компанія Dell використовує для всієї лінійки дискових накопичувачів PowerVault, лінійки серверів PowerEdge™ і зовнішніх дисків для робочих станцій Dell Precision™. Жорсткі диски SATA – це надійні й довговічні носії, що задовольняють всім галузевим стандартам зберігання даних. Передача файлів на PowerVault RD1000 займає приблизно стільки ж часу, скільки займає збереження даних на звичайному жорсткому диску ПК (швидкість до 80 Мбайт/с).

Тест на міцність касети при падінні:

Офіційно підтверджено, що касети RD1000 витримують падіння з висоти 1 м на бетонну підлогу, покрита плиткою. У більшості випадків касети RD1000 витримують падіння з висоти до 1,5 м.

Міцність і компактність

Касета PowerVault RD1000 працює в горизонтальному положенні або встановлена на боці. Завдяки компактній формі вона займає дуже мало місця на робочому столі або поруч із сервером, тому рішення для резервного копіювання

не вимагає перестановки меблів і встаткування в офісі. Знімна дискова касета відрізняється великою міцністю й легко міститься в портфелі, рюкзаку й навіть кишені пальто. Простої USB-Підключення PowerVault RD1000 за допомогою технології plug-and-play дозволяє створювати резервні копії декількох комп'ютерів і, якщо буде потрібно, взяти пристрій з будинку в офіс. Дискова касета важить приблизно стільки ж, скільки важить мобільний телефон, і лише трохи більше його по розмірі.

Дисковий пристрій для резервного копіювання Dell DR4100

Скоротите обсяги резервних копій за допомогою Dell DR4100, рішення нового покоління для резервного копіювання й відновлення на базі платформи Dell 12G:

– Оптимізуйте використання системи зберігання даних і доможіться зниження вимог до пропускної здатності глобальної мережі за допомогою потокової дедуплікації, стиску й дедуплікованої реплікації.

– Забезпечте високу продуктивність застосунків зі стиском і дедуплікацією на цільовій системі, що дозволить знизити вимоги до обчислювальної потужності сервера застосунків.

– Розгорніть рішення з урахуванням майбутніх потреб за допомогою системи DR4100 з поліпшеними функціями дедуплікації й стиску. Ці технології є найважливішими компонентами архітектури Dell Fluid Architecture, і вони інтегровані в усі продукти лінійки систем зберігання даних Dell.

Реплікація " трохи-до-одному" для аварійного відновлення

DR4100 пропонує економічний спосіб захистити кілька віддалених офісів за допомогою аварійного відновлення. Консолідація й оптимізації систем відновлення після збоїв і підтримка множинних процесів резервного копіювання за допомогою єдиного, простого в управлінні рішення:

– Забезпечення реплікації на один вузол з 32-х вузлів (макс.) одночасно, що дозволяє підвищити ефективність і скоротити витрати, пов'язані з наявністю декількох середовищ резервного копіювання.

– Надійний захист даних резервного копіювання за допомогою систем DR4000 і DR4100, що забезпечують реплікацію із шифруванням і стиском.

Гнучкі можливості розширення

Просте масштабування системи зберігання даних для резервного копіювання й відновлення відповідно до ваших бізнес-потреб. DR4100 поставляється в декількох конфігураціях, що відрізняються по ємності, що дозволяє розширювати систему в міру необхідності.

– DR4100 дозволяє здійснювати резервне копіювання більшого обсягу даних, чим може показувати її корисна ємність. Завдяки убудованим технологіям дедуплікації й стиску даних ви можете одержати до 27 Тбайт фізичної ємності й до 405 Тбайт логічної ємності в кожній системі.

– Підключите до двох полиць розширення обсягом 9, 18 або 27 Тбайт і ємність вашої системи буде рости разом з підприємством.

Низька сукупна вартість володіння й обслуговування

Впровадження готового, економічного, простого в обслуговуванні рішення для скорочення обсягів даних і їхнього захисту:

– Дисковий масив DR4100 має потужні убудовані функції й найкраще співвідношення ціни і якості в порівнянні із традиційними стрічковими накопичувачами для резервного копіювання.

– Скоротите час і витрати, пов'язані зі стрічковими системами резервного копіювання.

– Модель ліцензування "усе включено" DR4100 виключає непередбачені витрати й пропонує розширені функції й переваги без додаткових витрат.

Комплексна інтегрована підтримка програмного забезпечення

Впровадження рішення для резервного копіювання й відновлення даних, що підтримує використовуване вами програмне забезпечення, а також оптимізація роботи за допомогою інтеграції із сертифікованими застосунками:

– DR4100 підтримує лідируюче в галузі програмне забезпечення резервного копіювання й має безліч сертифікатів незалежних постачальників програмного забезпечення (ISV).

– Функція Rapid Data Access (швидкий доступ до даних) для DR інтегрується із програмним забезпеченням резервного копіювання, наприклад, застосунками Symantec OpenStorage (OST).

– Підтримка OST допомагає DR4100 забезпечувати швидке резервне копіювання й відновлення й надає розширені функції, такі як як реплікація на основі оптимізованої дуплікації.

Розширені засоби захисту даних

Захист критично важливих даних за допомогою засобів захисту, убудованих як в устаткування, так і програмне забезпечення DR4100. Надайте своєї організації більше ефективний захист від простоїв і аварійних ситуацій.

– DR4100 дозволяє захистити дані у випадку відключення живлення й здійснює превентивне виявлення перекручування даних у результаті апаратних збоїв.

– Підтримка конфігурацій RAID 6 дозволяє DR4100 забезпечити додатковий рівень захисту даних і витримати до двох одночасних відмов жорстких дисків без переривання роботи.

Спрощене, інтуїтивно зрозуміле керування

Розгортання готового рішення, призначеного для застосування в будь-якому середовищі резервного копіювання.

– Керування DR4100 здійснюється за допомогою інтуїтивно зрозумілого інтерфейсу у вигляді інформаційної панелі, що відображає стан системи, а також виконує автоматичний моніторинг робочого стану встаткування й перевірку цілісності ПЗ.

– Допоможіть знизити навантаження на IT-фахівців, здійснюючи щоденне швидке й повне резервне копіювання, що може виявитися більше ефективним, чим при використанні стрічкових накопичувачів.

– Підвищте ефективність роботи за допомогою повністю віддалені налаштування й адміністрування, а також спрощеного керування пулами дисків.

Time Machine

При першому резервному копіюванні Mac за допомогою Time Machine процедура може зайняти багато часу. Це пояснюється тим, що Time Machine зберігає більшість даних або всіх даних на комп'ютері Mac у резервну копію. Поки Time Machine виконує резервне копіювання у фоновому режимі, можна продовжувати роботу з комп'ютером Mac.

Після завершення резервного копіювання Time Machine продовжує роботу в тлі, копіюючи тільки файли, змінені з моменту останнього резервного копіювання. Це означає, що наступне резервне копіювання звичайно походить швидше.

Якщо потрібно призупинити резервне копіювання й завершити його пізніше, виберіть «Пропустити цю копію» у меню Time Machine. Time Machine автоматично спробує продовжити резервне копіювання пізніше. Якщо потрібно почати резервне копіювання вручну, виберіть у меню команду «Створити резервну копію зараз».

Резервне копіювання великого обсягу змін

Іноді на резервне копіювання може вимагатися більше часу, ніж звичайно, якщо з моменту останнього копіювання внесені зміни в багато файлів або в більші файли. У таких випадках повідомлення «Підготовка» у меню Time Machine відображається довше, ніж звичайно.

Приклади:

– Коли диск резервного копіювання недоступний (наприклад, у поїздці або коли диск резервного копіювання не приєднаний або відключений від живлення), Time Machine не може виконувати резервне копіювання файлів. Якщо комп'ютер Mac використовується протягом декількох днів без доступу до диска резервного копіювання, то після поновлення доступу резервне копіювання займе більше часу.

– Якщо на комп'ютері використовується ПЗ віртуалізації, таке як Parallels або VMWare, цим ПЗ може бути створений великий файл образа диска або інший файл для зберігання даних, пов'язаних з іншими операційними системами. Time Machine може спробувати зберегти резервну копію всього образа диска, навіть якщо в ньому змінилося лише кілька файлів. Щоб забезпечити оптимальні результати, обновите

програмне забезпечення, потім перейдіть на сайт підтримки розроблювача, щоб знайти інформацію про використання Time Machine із цими програмами. Якщо потрібно пропускати ці файли при резервному копіюванні Time Machine, їх можна виключити з резервної копії в налаштуваннях Time Machine.

– Якщо недавно встановлювалося нове програмне забезпечення або обновлялася macOS, на завершення наступного резервного копіювання може знадобитися більше часу. Коли Time Machine завершить обробку нового програмного забезпечення, наступне резервне копіювання буде проходити швидше.

– Якщо попереднє резервне копіювання було скасовано або зненацька перерване, наступного разу Time Machine може витратити більше часу на обробку файлів. Таке також можливо, якщо робота комп'ютера Mac була завершена неправильно або якщо диск не був витягнутий перед його відключенням.

Швидкість мережного підключення

Якщо резервне копіювання виконується по мережі Wi-Fi, переконаєтеся, що крапка доступу Wi-Fi або маршрутизатор перебуває досить близько. Підключення до мережі Wi-Fi може вповільнюватися, якщо комп'ютер Mac перебуває занадто далеко від маршрутизатора. Щоб одержати подання про силу сигналу бездротової мережі, подивитеся на меню Wi-Fi. Якщо необхідно, перевірте наявність проблем з підключенням Wi-Fi.

Антивірусне програмне забезпечення

Якщо на комп'ютері Mac установлене антивірусне програмне забезпечення, переконаєтеся, що воно оновлено до останньої версії. Якщо воно заважає резервному копіюванню комп'ютера, спробуйте виключити диск резервного копіювання з перевірки на віруси. Для одержання додаткової інформації перевірте документацію по антивірусній утиліті або звернетеся до її розроблювача.

Перевірка дисків

Резервне копіювання може виконуватися повільніше, якщо виникає проблема з одним з копіюємих дисків або з диском, на якому зберігаються копії.

Якщо використовується пристрій AirPort Time Capsule, на ньому можна настроїти перевірку убудованого диска.

– Відключите AirPort Time Capsule від джерела змінного струму.

– Почекайте десять секунд, потім знову підключите AirPort Time Capsule до джерела живлення.

– Коли убудований диск Time Capsule працює, світлодіодний індикатор на пристрої стає зеленим. При наявності проблем індикатор мигає жовтим. Відкрийте Утиліту AirPort і підключите Time Capsule, щоб одержати додаткові відомості про проблему. Якщо усунути проблему не вдається, може знадобитися видалити дані з жорсткого диска за допомогою Утиліти AirPort після створення додаткової резервної копії за допомогою Time Machine і іншого диска. Якщо видалити дані з диска не вдається, пристрій Time Capsule може мати потребу в обслуговуванні.

Для інших дисків відкрийте налаштування Time Machine і виключите Time Machine, потім скористайтеся Дисковою утилітою для перевірки завантажувального диска, зовнішніх дисків резервного копіювання й будь-яких інших дисків, резервне копіювання яких виконується. Можна знову включити Time Machine після успішної перевірки або відновлення дисків.

All Backup

All Backup реалізований на базі **ПЗ CommVault Simpana** – визнаного лідера в області систем резервного копіювання й архівного зберігання корпоративного рівня.

Керуючий сервер і портал самообслуговування розміщені в Хмарі De Novo. У інфраструктурі, яка захищається, установлюються сервери зберігання даних і агенти, що забезпечують інтеграцію з віртуальної й/або фізичною інфраструктурою, а також прикладним програмним забезпеченням.

Обмін командами й звітами між керуючим сервером і серверами зберігання даних відбувається через Інтернет по захищеному каналі SSL/TLS. Також, можливе використання виділених каналів у випадку зберігання резервних копій на площадці De Novo.

Щомісячна вартість послуги визначається кількістю й типом об'єктів, що захищаються, що може гнучко мінятися відповідно до потреб, як у більшу, так і в меншу сторону.

Широкі можливості All Backup

Захист різнорідних об'єктів. Захищайте об'єкти віртуальної інфраструктури (віртуальні машини), файлові системи віртуальних і фізичних серверів, а також дані застосунків з урахуванням їх специфіки. Підтримка широкого спектра:

– процесорних архітектур (x86, POWER, SPARC);

– операційних систем (Windows, Linux, FreeBSD, AIX, HP-UX, Solaris, MacOS);

– застосунків (MS Active Directory, IBM Notes або IBM Domino, MS Exchange Server, MS SharePoint Server і ін.) $

– СУБД (MS SQL Server, PostgreSQL, Oracle, DB2, Informix, MySQL, SAP, Sybase і ін.).

Інтеграція з різними системами віртуалізації: VMware, Microsft Azure, Microsoft Hyper-V, Amazon Web Services, OpenStack.

Захист на будь-якій інфраструктурі. Об'єкти, що захищаються, можуть перебувати як у власній інфраструктурі, так і в хмарі.

Зберігання резервних копій де завгодно

– Можливість використання як локальних, так і віддалених репозиторіїв різних типів (блокові пристрої, мережні диски, об'єктне сховище S3/Swift), у тому числі комбінацій «локальний + віддалений».

– Можливість зберігання резервних копій у хмарах De Novo, Amazon Web Services, Microsoft Azure.

Безпечне й ефективне використання репозитарія. Дедуплікація, компресія й криптозахист резервних копій забезпечують безпеку й високу ефективність використання репозитарія резервних копій.

Портал самообслуговування. Дозволяє гнучко управляти розкладами й гранулярністю резервного копіювання й відновлення, контролювати статус завдань резервного копіювання й переглядати статистичну й аналітичну інформацію.

Переваги резервного копіювання за допомогою All Backup

Легкий і швидкий старт

– Підключення протягом 1-2 робочих днів. Доступний тестовий період.

Гнучкість

– У будь-який момент користувач може змінити обсяг споживання сервісу й підключити нові об'єкти до системи резервного копіювання або ж відключити неактуальні.

Відсутність ризиків капітальних витрат

– На відміну від впровадження власної системи резервного копіювання, капітальні витрати, проектні ризики й безповоротні втрати відсутні.

Безпека корпоративного рівня

– Дані, що захищаються, залишаються в периметрі корпоративної мережі замовника.

– Підключення до сервісу тільки по захищеному каналі SSL/TLS.

– Керуючий сервер і портал самообслуговування розміщений у Хмарі De Novo, що побудовано за моделю Trusted Cloud і забезпечено максимальним захистом від зовнішніх погроз.

Відповідальність за доступність сервісу

– De Novo гарантує доступність сервісу 99.5%. Фінансова відповідальність за виконання SLA.

Моделі використання

Локальне резервне копіювання – найбільше що часто використовується вид резервного копіювання, що дозволяє захиститися від базового набору погроз (не пов'язаних з виходом з ладу площадки цілком). Локальне резервне копіювання має на увазі зберігання РК на тій же площадці де розміщені й

продуктивні ВМ, але на іншій системі зберігання даних. Резервування й відновлення відбуваються в рамках однієї площадки, що забезпечує максимальну швидкодію

Disaster Recovery copy вид резервного копіювання має на увазі зберігання РК на віддалених сховищах, що дозволяє відновлюватися навіть у випадку збоїти локальної площадки цілком (звичайно використовуються хмарні сховища даних як наприклад S3)

Комбінація локального резервного копіювання й Disaster Recovery copy дозволяє об'єднати максимальний захист і швидкість.

Крос гіпервізорне відновлення (Cross Hypervisor Restore). Незважаючи на те що завдання міграції з одного гіпервізора на іншій (наприклад, з Hyper-V на VMware ESXi) виникає досить рідко, вона може викликати більші проблеми особливо коли мігруємих ВМ багато. Послуга All Backup дозволяє резервну копію, зроблену на одному типі гіпервізора відновити на іншому гіпервізорі.

Сховище дані спостереження

Дані, які необхідні для спостереження за працездатністю застосунка або для усунення неполадок, пов'язаних з роботою застосунків, зберігаються в сховище дані спостереження. У цьому розділі втримується опис відслідковуваємих даних і зв'язаних запитів, а також командлетів Windows PowerShell, призначених для спостереження.

Дані, які відслідковуються, в базі даних спостереження

До появи AppFabric було важко усувати неполадки, пов'язані з застосунком, розподіленим по декількох службах і комп'ютерам. Необхідно було включити трасування кожної служби, виконувати певні дії, відключати трасування, збирати всі файли трасування, а потім переглядати їх у засобі перегляду трасування служб.

В AppFabric можна використовувати сховище дані спостереження разом з однієї або декількома екземплярами Event Collection service для спрощення усунення неполадок, пов'язаних з застосунками. Всі події служб WCF і

робітників процесів зберігаються в сховище дані спостереження. Дані подій інструментування для різних служб WCF і робітників процесів можуть зберігатися в одному сховищі дані спостереження. Також для кожного застосунка можна виділити окреме сховище.

Дані подій інструментування, що зберігаються в сховище дані спостереження, складаються з наступних елементів:

– Події трасування WCF у режимі реального часу, призначені для спостереження, усунення неполадок і налаштування служб WCF. У цю категорію входять події трасування (передачі), які дозволяють реконструювати потік повідомлень між службами.

– Події із записів відстеження WF, що співвідносяться по ідентифікаторі екземпляра.

– Події вузла служби. Ця категорія містить будь-які події, що відбулися на вузлі служб WCF або робітників процесів.

Нерідко розширення WCF і налаштування прив'язки міняють послідовність повідомлень служби WCF. Внаслідок цього в сховище дані спостереження можуть з'явитися події, які не були створені в рамках користувальницької бізнесу-логіки. Наприклад, стійкі робочі процеси, керовані службою керування робочими процесами, надають кінцеву крапку керування, що дозволяє цій службі передавати команди іншій службі за допомогою повідомлень net.pipe. При обробці цих повідомлень створюються події, які обробляються в такий же спосіб, що й бізнесу-повідомлення. Це буде відбуватися при кожному одержанні або відправленні повідомлення веб-службою.

Запити, підтримувані базою дані спостереження

Сховище дані спостереження підтримує запити для великої кількості метрик служб. Деякі з доступних можливостей наведені нижче.

Метадані служби: ім'я комп'ютера, ім'я сайту, ім'я застосунка й віртуальний шлях служби. Загальні параметри введення для служб включають метадані служби, ім'я метрики й діапазон часу.

– Запити подій WF, пов'язані з даним ідентифікатором екземпляра WF, станом події або типом події.

– Запит по метаданим служби, наприклад: число екземплярів, запущених у певний період часу, активні екземпляри в певний період часу, середній строк життя екземпляра в певний період часу.

– Запит змінних і користувальницьких даних, що відслідковуються, для екземпляра або служби, заснований на ідентифікаторі події запису відстеження, ідентифікаторі екземпляра WF або метаданих служби.

– Запит метрик служби WCF по типі події аналітичного трасування WCF.

– Запит на реконструкцію потоку повідомлень для зазначеного ідентифікатора дії.

– Запит, заснований на користувальницьких даних і змінних, витягнутих з екземплярів служби робочих процесів.

– Запит подій WCF, пов'язаних з даним екземпляром служби робочих процесів.

Схема бази даних

Схема спостереження ставиться до всіх об'єктів, які створюються при ініціалізації командлетом Initialize-**ASMonitoringDatabase** нового сховища дані спостереження. Не підтримується зміна або розширення користувачами об'єктів у схемі спостереження.

Можна виконувати SQL-Запити прямо, звертаючись до сховища дані спостереження, що дозволить переглядати дані подій.

Схема сховища дані спостереження може існувати спільно зі схемою сховища зберігаємість в одному сховищі.

Командлети бази дані спостереження

Сховище дані спостереження управляється за допомогою командлетів AppFabric. Командлети використовуються в наступних областях.

Керування базою даних

Initialize-ASMonitoringDatabase

Створює й ініціалізує нове сховище дані спостереження.

Remove-ASMonitoringDatabase

Видаляє сховище дані спостереження. Якщо сховище є виділеним сховищем, воно віддаляється. Якщо сховище використовується разом з іншою схемою, віддаляється тільки схема спостереження.

Очищення й архівація

Використовуйте цю групу командлетів для керування налаштуваннями архівації сховища дані спостереження.

Clear-ASMonitoringDatabase

Очищає зазначені дані в сховище дані спостереження. Дані можуть бути відправлені в раніше настроєне сховище архівів (за допомогою командлета Set-MonitoringDatabaseArchiveConfiguration).

Set-ASMonitoringDatabaseArchiveConfiguration

Зв'язує сховище дані спостереження із цільовим архівним сховищем. Цільове архівне сховище повинне бути попередньо створеним і инициализированним сховищем дані спостереження.

Get-ASMonitoringDatabaseArchiveConfiguration

Одержує відомості про конфігурацію для архівного сховища зазначеного сховища дані спостереження. Конфігурація повинна бути раніше визначена за допомогою командлета Set-MonitoringDatabaseArchiveConfiguration.

Remove-ASMonitoringDatabaseArchiveConfiguration

Видаляє відомості про конфігурацію, що ставляться до архівного сховища сховища дані спостереження.

AppFabric 1.1

AppFabric зберігає відомості про зберігаємість й дані спостереження в базі даних. При виборі параметрів конфігурації за замовчуванням у процесі установки будуть створені два сховища даних SQL Server, ApplicationServerMonitoring і

ApplicationServerPersistence, які будуть використовуватися відповідно для зберігання відомостей про зберігаємість й дані спостереження. Проте можна настроїти додаткові сховища даних при необхідності масштабування або поділу даних за допомогою засобу конфігурації або командлетів PowerShell, що поставляються разом з AppFabric.

Облікові записи для входу в базу даних

При використанні SQL Server AppFabric створює групи Windows і зіставляє їх з ролями бази даних SQL Server. Ці ролі використовуються для керування доступом до даних у сховище даних, тому тільки користувачі, що є членами відповідної групи Windows, мають доступ до даних.

ArcGIS Data Store

Ви можете використовувати Майстер налаштування ArcGIS Data Store для створення сховища реляційних даних і кешу і його підключення до сайту ArcGIS Server.

Або, якщо у вас немає доступу до веб-браузера або ви хочете створити велике часове-просторово-тимчасове сховище даних, запустите утиліту configuredatastore на комп'ютері, де встановлений ArcGIS Data Store.

Сайт ArcGIS Server може мати тільки одну керовану базу даних. Якщо ви хочете створити реляційне сховище даних, але ваш сайт ArcGIS Server уже має керовану базу даних, ви можете виконати одне із трьох дій перед тим, як створювати реляційне сховище даних:

– Видалити керовану базу даних і додати її знову, як зареєстровану базу даних. У цьому випадку ваші існуючі розміщені векторні шари будуть продовжувати працювати. Коли ви приєднаєте ArcGIS Data Store до сайту ArcGIS Server, дані нових опублікованих розміщених векторних шарів і кеш шарів 3D сцен будуть записуватися в сховище даних. Цей найпростіший рекомендувати^ся метод, що.

– Видалити керовану базу даних Прибігайте до цього методу, тільки якщо вам не потрібні існуючі розміщені векторні шари, оскільки вони перестануть

працювати, якщо ви видалити керовану базу даних і не додасте її як зареєстрована база даних.

– Якщо ви не можете залишити існуючу керовану базу даних, виконаєте експорт даних існуючих розміщених векторних шарів, видалите існуючі векторні шари, видалите базу даних, з'єднаєте ArcGIS Data Store зі своїм сайтом ArcGIS Server і опублікуйте експортовані дані заново. Більше докладну інформацію див. у розділі Перехід на ArcGIS Data Store.

Використовуйте майстер Налаштування сховища даних для створення реляційного сховища даних і сховища даних кеша аркушів.

Якщо ви запускаєте майстер установки, Майстер конфігурації Data Store відкриється у веб-браузері за замовчуванням. Або можна відкрити майстер за допомогою ярлика на комп'ютері.

– Укажіть URL-адресу сайту ArcGIS Server, для якого ви створюєте й реєструєте сховище даних.

URL-адреса має вигляд https://gisserver.domain.com:6443. Зверніть увагу, навіть якщо ваш сайт ArcGIS Server використовує веб-адаптер, URL-адреса повинен бути уведений у зазначеному вище форматі.

1. Уведіть ім'я користувача й пароль адміністратора ArcGIS Server і клацніть **Далі**.

Користувач повинен бути убудованим (не корпоративним).

– Уведіть місце розташування директорії сховища даних.

Ця директорія містить файли зберігання даних і директорію з архівами.

Ви не можете використовувати шлях UNC при завданні директорії файлів сховища даних. Тримаєте файли сховища даних на тім же комп'ютері, де встановлений ArcGIS Data Store. Однак директорія архівів містить файли архівів, необхідні для відновлення даних опублікованих шарів об'єктів. Після завершення налаштування сховища даних, перемістите архівну директорію на мережний диск іншого комп'ютера, не того, котрий ви вибрали для установки й зберігання даних.

При переході до архівних папок можна використовувати шлях UNC. Застосункову інформацію див. у розділі Керування архівами сховища даних.

2. Клацніть **Далі**.

3. Переглянете інформацію на закладці **Підсумкова інформація про конфігурацію**. Якщо все правильно, клацніть **Завершити**. Якщо необхідно внести зміни, клацніть **Назад**.

Помнете, що перше сховище даних, зареєстроване на сайті ArcGIS Server, є первинним комп'ютером. Якщо ви додасте ще одне сховище даних на той же сайт ArcGIS Server, воно буде черговим комп'ютером.

По завершенні установки ви одержуєте первинний комп'ютер сховища даних, що використовується сайтом ArcGIS Server для зберігання даних опублікованих шарів об'єктів і кешів шарів 3D сцен, опублікованих на порталі.

Якщо при установці з'являлися помилки, клацніть **Далі** в діалоговому вікні **Підсумкова інформація про конфігурацію**, щоб знову відкрити Майстер конфігурації Data Store і задати інші відомості. Про причини помилок і методи їхнього усунення див. Усунення неполадок ArcGIS Data Store.

Тепер ви можете задати місце розташування архіву.

Використання утиліти configuredatastore

Можна використовувати командну утиліту configuredatastore для створення будь-якого типу сховища даних і його реєстрації на сайті ArcGIS Server, її необхідно використовувати, якщо потрібно створити часове-просторово-тимчасове сховище більших даних або сховище даних кеша без сховища реляційних даних. Утиліта встановлена в <ArcGIS Data Store installation directory>/tools.

1. Відкрийте Командний рядок, використовуючи опцію **Запустити від імені адміністратора**.

– Запустите файл configuredatastore.bat і вкажіть URL-адресу ArcGIS Server, на якому ви хочете зареєструвати сховище даних, ім'я користувача й

пароль адміністратора ArcGIS Server і шлях до директорії даних ArcGIS Data Store, уведіть сховище даних, що будете створювати.

У цьому прикладі URL-адреса ArcGIS Server – https://myserver.domain.com:6443/arcgis/admin, ім'я адміністратора й пароль – siteadmin і T1n@sp, а директорія даних – C:\data\, створюється сховище реляційних даних.

```
configuredatastore https://myserver.domain.com:6443/arcgis/admin siteadmin T1n@sp c:\data\ ---istores relational
```

Зверніть увагу, що при створенні сховища реляційних даних також створюється сховище даних кеша, однак воно не запущено, поки не опублікований перший розміщений шар сцени.

Якщо необхідно створити кілька типів сховищ даних на одній машині в одній директорії, можна вказати значення, розділені комами, однак це не рекомендується. Наприклад, можна створити сховище даних кеша й часових-просторово-тимчасових більших даних, указавши tileCache,spatiotemporal з опцією stores.

Створюється первинний комп'ютер сховища даних і реєструється на сайті ArcGIS Server.

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 першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи оцінки продуктивності систем зберігання даних.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Підготовку нового сервера до роботи варто починати з налаштування резервного копіювання. Всі, здавалося б, про це знають – але часом навіть досвідчені системні адміністратори допускають непрощенні помилки. І справа тут не тільки в тому, що завдання налаштування нового сервера потрібно вирішувати дуже швидко, але ще й у тому, що далеко не завжди буває ясно, який спосіб резервного копіювання потрібно використовувати.

Звичайно, ідеальний спосіб, який би всіх улаштовував, створити неможливо: скрізь є свої плюси й мінуси. Але в той же час цілком реальним представляється підібрати спосіб, що максимально підходить під специфіку конкретно проекту.

При виборі способу резервного копіювання потрібно насамперед звернути увагу на наступні критерії:

– Швидкість (час) резервного копіювання в сховище;

– Швидкість (час) відновлення з резервної копії;

– Скільки копій можна буде тримати при обмеженому розмірі сховища (сервері зберігання бекапів);

– Обсяг ризиків через неконсистентності резервні копії, неналагодженості методу виконання бекапів, повної або часткової втрати бекапів;

– Накладні витрати: рівень навантаження, створюваної на сервер при виконанні копіювання, зменшення швидкості відгуку сервісу й т.п.

– Вартість оренди всіх сервісів, що використовуються.

У цьому розділі розповімо про основні способи резервного копіювання серверів під керуванням Linux-систем і про найбільш типові проблеми, з якими можуть зштовхнутися новачки в цій дуже важливій області системного адміністрування.

Схема організації зберігання й відновлення з резервних копій

При виборі схеми організації методу резервування варто звернути увагу на наступні базові моменти:

– Резервні копії не можна зберігати в одному місці з резервуємими даними. Якщо ви зберігаєте резервну копію на одному дисковому масиві з вашими даними, то ви втратите її у випадку ушкодження основного дискового масиву.

– Дзеркальовані (RAID1) не можна порівнювати з резервним копіюванням. Рейд захищає вас тільки від апаратної проблеми з одним з дисків (а рано або пізно така проблема буде, тому що дискова підсистема майже завжди є вузьким місцем на сервері). До того ж при використанні апаратних рейдів є ризик поломки контролера, тобто необхідно зберігати його запасну модель.

– Якщо ви зберігаєте резервні копії в рамках однієї стійки в дата-центрі (ДЦ) або просто в рамках одного ДЦ, то в такій ситуації теж є певні ризики.

– Якщо ви зберігаєте резервні копії в різних ДЦ, то різко зростають витрати на мережу й швидкість відновлення з віддаленої копії.

Часто причиною відновлення даних служить ушкодження файлової системи або дисків. Тобто бекапи потрібно зберігати десь на окремому сервері-сховищі. У цьому випадку проблемою може стати «ширина» каналу передачі даних. Якщо у вас виділений сервер, то резервне копіювання дуже бажано виконувати по окремому мережному інтерфейсі, а не на тім же, що виконує обмін даних із клієнтами. Інакше запити вашого клієнта можуть не «поміститися» в обмежений канал зв'язку. Або через трафіку клієнтів бекапи не будуть зроблені в строк.

Далі потрібно подумати про схему й час відновлення даних з погляду зберігання бекапів. Може бути вас цілком улаштовує, що бекап виконується за 6 годин уночі на сховище з обмеженою швидкістю доступу, однак відновлення довжиною в 6 годин вас навряд чи влаштує. Значить доступ до резервних копій повинен бути зручним і дані повинні копіюватися досить швидко. Так,

наприклад, відновлення 1Тб даних зі смугою в 1Гб/с займе майже 3 години, і це якщо ви не «упретеся» у продуктивність дискової підсистеми в сховищі й сервері. І не забудьте додати до цього час виявлення проблеми, час на рішення про відкіт, час перевірки цілісності відновлених даних і обсяг наступного невдоволення клієнтів/колег.

Інкрементальне резервне копіювання

При **інкрементальному** резервному копіюванні копіюються тільки файли, які були змінені із часу попереднього бекапа. Наступне інкрементальне резервне копіювання додає тільки файли, які були змінені з моменту попереднього. У середньому інкрементальне резервне копіювання займає менше часу, тому що копіюється менша кількість файлів. Однак процес відновлення даних займає більше часу, тому що повинні бути відновлені дані останнього повного резервного копіювання, плюс дані всіх наступних інкрементальних резервних копіювань. При цьому на відміну від диференціального копіювання, що змінилися або нові файли не заміщають старі, а додаються на носій незалежно.

Інкрементальне копіювання найчастіше виробляється за допомогою утиліти rsync. З його допомогою можна заощадити місце в сховище, якщо кількість змін за день не дуже велико. Якщо змінені файли мають великий розмір, то вони будуть скопійовані повністю без заміни попередніх версій.

Процес резервного копіювання за допомогою rsync можна розділити на наступні кроки:

– Складається список файлів на резервуємому сервері й у сховище, по кожному файлі зчитуються метадані (права, час зміни й т.д.) або контрольна сума (при використанні ключа -checksum).

– Якщо метадані файлів відрізняються, то файл б'ється на блоки й по кожному блоці вважається контрольна сума. блоки, Що Відрізняються, накачуються в сховище.

– Якщо під час підрахунку контрольних сум або передачі файлу в нього була внесена зміна, його резервування повторюється з початку.

– За замовчуванням rsync передає дані через SSH, а значить кожний блок даних додатково шифрується. Rsync можна також запустити як демон і передавати дані без шифрування по його протоколі.

З більше докладною інформацією про роботу rsync можна ознайомитися на офіційному сайті.

Для кожного файлу rsync виконує дуже велика кількість операцій. Якщо файлів на сервері багато або якщо процесор сильно завантажений, то швидкість резервного копіювання буде істотно знижена.

З досвіду можемо сказати, що проблеми на SATA-дисках (RAID1) починаються приблизно після 200G даних на сервері. Насправді всі, кінцеве ж, залежить від кількості inode. І в кожному випадку ця величина може зміщатися як в одну так і в іншу сторону.

Після певної риси час виконання резервного копіювання буде дуже довгим або попросту не буде відпрацьовувати за добу.

Для того, щоб не порівнювати всі файли, є lsyncd. Цей демон збирає інформацію про файли, що змінилися, тобто ми вже заздалегідь будемо мати готовий їхній список для rsync. Треба, однак, урахувати, що він дає додаткове навантаження на дискову підсистему.

Диференціальне резервне копіювання

При **диференціальному** резервному копіюванні кожний файл, що був змінений з моменту останнього повного резервного копіювання, копіюється щораз заново. Диференціальне копіювання прискорює процес відновлення. Усе, що вам необхідно – це остання повна й остання диференціальна резервна копія. Популярність диференціального резервного копіювання росте, тому що всі копії файлів робляться в певні моменти часу, що, наприклад, дуже важливо при зараженні вірусами.

Диференціальне резервне копіювання здійснюється, наприклад, за допомогою такої утиліти, як rdiff-backup. При роботі із цією утилітою виникають ті ж проблеми, що й при інкрементальному резервному копіюванні.

У цілому, якщо при пошуку різниці в даних здійснюється повний перебір файлів, проблеми такого роду резервування аналогічні проблемам з rsync.

Хочемо окремо відзначити, що якщо у вашій схемі резервного копіювання кожний файл копіюється окремо, те варто видаляти/виключати непотрібні вам файли. Наприклад, це можуть бути кеши CMS. У таких кешах звичайно дуже багато маленьких файлів, втрата яких не позначиться на коректній роботі сервера.

Повне резервне копіювання

Повне копіювання звичайно торкає всієї вашої системи й всіх файлів. Щотижневе, щомісячне й щоквартальне резервне копіювання має на увазі створення повної копії всіх даних. Звичайно воно виконується по п'ятницях або протягом вихідних, коли копіювання великого обсягу даних не впливає на роботу організації. Наступні резервні копіювання, виконувані з понеділка по четвер до наступного повного копіювання, можуть бути диференціальними або інкрементальними, головним чином для того, щоб зберегти час і місце на носії. Повне резервне копіювання варто проводити принаймні щотижня.

У більшості публікацій по відповідній тематиці рекомендується повне резервне копіювання виконувати один або два рази в тиждень, а в інший час час – використовувати інкрементальне й диференціальне. У таких радах є свій резон. У більшості випадків повного резервного копіювання раз у тиждень цілком достатньо. Виконувати його повторно має сенс у тому випадку, якщо у вас немає можливості на стороні сховища актуалізувати повний бекап і для забезпечення гарантії коректності резервної копії (це може знадобитися, наприклад, у випадках, якщо ви по тим або інших причинах не довіряєте наявним у вас скриптам або софту для резервного копіювання.

Насправді повне резервне копіювання можна поділити на 2 частині:

– Повне резервне копіювання на рівні файлової системи;

– Повне резервне копіювання на рівні пристроїв.

Розглянемо їхні характерні риси на прикладі:

```
root@komarov:~# df -h
Filesystem                         Size  Used Avail Use% Mounted on
/dev/mapper/komarov_ system-root  3.4G  808M  2.4G  25% /
/dev/mapper/komarov_ system-home  931G  439G  493G  48% /home
udev                              383M  4.0K  383M   1% /dev
tmpfs                             107M  104K  107M   1% /run
tmpfs                             531M     0  531M   0% /tmp
none                              5.0M     0  5.0M   0% /run/lock
none                              531M     0  531M   0% /run/shm
/dev/xvda1                        138M   22M  109M  17% /boot
```

Резервувати ми будемо тільки /home. Все інше можна швидко відновити вручну. Можна також розгорнути сервер системою керування конфігураціями й підключити до нього наш /home.

Повне резервне копіювання на рівні файлової системи

Типовий представник: dump.

Утиліта створює «дамп» файлової системи. Можна створювати не тільки повну, але й інкрементальну резервну копію. dump працює з таблицею inode і «розуміє» структуру файлів (так, розріджені файли стискуються).

Створювати дамп працюючої файлової системи «нерозумно й небезпечно», тому що файлова система (ФС) може змінюватися під час створення дампа. Його треба створювати зі снапшоту (трохи пізніше ми обговоримо особливості роботи зі снапшотами більш докладно), відмонтованої або замороженої ФС.

Така схема так само залежить від кількості файлів, і час її виконання буде рости з ростом кількості даних на диску. У той же час в dump швидкість роботи вище, ніж в rsync.

У випадку, якщо потрібно відновити не резервну копію цілком, а, наприклад, тільки пари випадково зіпсованих файлів), добування таких файлів утилітою restore може зайняти занадто багато часу.

Повне резервне копіювання на рівні пристроїв

mdraid і DRBD

Фактично настроюється RAID1 з диском/рейдом на сервері й мережному диску, і час від часу (по частоті виконання бекапів) додатковий диск синхронізується з основним диском/рейдом на сервері.

Найбільший плюс – швидкість. Тривалість виконання синхронізації залежить тільки від кількості внесених за останній день змін.

Така система резервного копіювання використовується досить часто, але мало хто усвідомлює тім, що отримані з її допомогою резервні копії можуть бути недієздатними, і от чому. Коли синхронізація дисків завершена, диск із резервною копією відключається. Якщо в нас, наприклад, запущена СУБД, що пише дані на локальний диск порціями, зберігаючи проміжні дані в кеші, немає ніякої гарантії того, що вони взагалі потраплять на бекапний диск. У найкращому разі ми втратимо частину змінюваних даних. Тому такі бекапи навряд чи варто вважати надійними.

LVM + dd

Снапшоти – чудовий инстумент для створення консистентних бекапів. Перед створенням снапшота необхідно скинути кеш ФС і вашого ПЗ на дискову підсистему.

Наприклад, з одним MySQL це буде виглядати так:

```
$ sudo mysql -e 'FLUSH TABLES WITH READ LOCK;'
$ sudo mysql -e 'FLUSH LOGS;'
$ sudo sync
$ sudo lvcreate -s -p r -l100%free -n %s_backup /dev/vg/%s
$ sudo mysql -e 'UNLOCK TABLES;'
```

Далі можна копіювати снапшот у сховище. Головне – стежити за тим, щоб під час копіювання снапшот не самознищився й не забувати, що при створенні снапшота швидкість запису впаде в рази.

Бекапи СУБД можна створити окремо (наприклад, використовуючи бінарні логи), усунувши тим найпростіший на час скидання кеша. А можна

створювати дампи в сховище, запустивши там инстанс СУБД. Резервне копіювання різних СУБД – це тема для окремих публікацій.

Копіювати снапшот можна з використанням докачки (наприклад, rsync з патчем для копіювання блокових пристроїв:

bugzilla.redhat.com/show_bug.cgi?id=494313),

можна по блоках і без шифрування (netcat, ftp). Можна передавати блоки в стислому виді й монтувати їх у сховище за допомогою AVFS, і примонтувати на сервері розділ з бекапами по SMB.

Стиск усуває проблеми швидкості передачі, завантаження каналу й місця в сховище. Але, однак якщо ви не використовуєте AVFS у сховище, то на відновлення тільки частини даних у вас піде багато часу. Якщо будете використовувати AVFS, то зштовхнетеся з її «вогкістю».

Альтернатива стиску блоками – squashfs: можна підмонтувати, приміром, по Samba розділ до сервера й виконати mksquashfs, але ця утиліта так само працює з файлами, тобто залежить від їхньої кількості.

До того ж при створенні squashfs витрачається досить багато ОЗП, що може легко привести до виклику oom-killer.

Безпека

Необхідно убезпечити себе від ситуації коли сховище або ваш сервер будуть зламані. Якщо зламано сервер, то краще щоб не було прав на видалення/зміна файлів у сховище в користувача, що записує туди дані.

Якщо зламано сховище, то права бекапного користувача на сервері так само бажано обмежити по максимуму.

Якщо канал резервного копіювання може бути прослуханий, то потрібні засоби шифрування.

У підсумку, при виборі системи резервного копіювання під ваш проект, потрібно провести тести обраного типу резервного копіювання й звернути увагу на:

– час резервного копіювання в поточній стадії проекту;

– час резервного копіювання у випадку, якщо даних буде в рази більше;

– навантаження на канал;
– навантаження на дискову підсистему на сервері й у сховище;
– час відновлення всіх даних;
– час відновлення пари файлів;
– необхідність у консистентності даних, особливо БД;
– витрата пам'яті й наявність викликів oom-killer;

3.2 Розробка структурної схеми

Продуктивність IT-системи в цілому визначається продуктивністю її окремих компонентів:

– за стосунків;
– операційної системи;
– фізичного або віртуального сервера;
– мережі передачі даних між сервером і СЗД;
– самої СЗД.

Кожний із цих компонентів, як правило, складається з безлічі окремих підсистем, кожна з яких здатна впливати на загальну продуктивність IT-системи. Так, недолік оперативної пам'яті в сервері навіть при наявності 100 високопродуктивних процесорних ядер може привести до помітного зниження загальної продуктивності IT-системи. Або, приміром, невірно обрана конфігурація дискової підсистеми СЗД старшого класу буде «гальмувати» всю IT-систему, незважаючи на те що така СЗД здатно витримати дуже більше навантаження.

Очевидно, що продуктивність IT-системи – це не сума показників всіх її компонентів. Найчастіше вона залежить від характеристик найбільше «слабкої ланки». Тому якщо ставиться завдання підвищити продуктивність IT-системи, те не завжди необхідно купувати нову СЗД, установлювати нові сервери із процесорами останнього покоління й т.п. Набагато важливіше виявити вузьке

місце в поточній конфігурації ІТ-системи й зрозуміти, чи можна виправити ситуацію, використовуючи наявні ресурси. Наприклад, заміна СЗД на флеш-масив (All-Flash Array, AFA) може дати приріст продуктивності ІТ-системи на кілька десятків відсотків, тоді як проста оптимізація SQL-Запиту до СУБД дозволить збільшити цю продуктивність багаторазово.

Якщо ж покупка нової СЗД неминуча, потрібно ретельно продумати всі деталі. Щоб правильно вибрати нову СЗД або спланувати модернізацію існуючої, необхідно зібрати всі вимоги, пропоновані ІТ-системою до зберігання даних, основні з яких – ємність і продуктивність. І якщо питання «Скільки терабайтів корисної ємності вам потрібно?», як правило, не викликає проблем, то прохання вказати величину необхідної продуктивності можуть багатьох загнати в глухий кут.

Здавалося б, простіше нікуди: щоб забезпечити високу продуктивність ИТ для всіх підрозділів бізнесу, потрібно купити саму дорогу й швидку СЗД – тільки й усього. Але так може здатися тільки на перший погляд. Таким придбанням будуть задоволені далеко не всі відділи підприємства, і насамперед, звичайно, фінансовий департамент, адже між продуктивністю СЗД і її вартістю існує чітка кореляція, а витрати на впровадження повинні якнайменше позначатися на бюджеті. Разом з тим чим менше засобів буде витрачено на нову СЗД, тим нижче буде її продуктивність – і тоді в збитку виявляться рядові користувачі ІТ-системи. Відповідно, ІТ-фахівець, що відповідає за впровадження нового сховища даних, повинен знайти золоту середину, коли й фінансові обмеження будуть враховані, і продуктивність виявиться достатньою.

Для вже існуючих ІТ-систем, яким потрібно лише відновлення (приміром, у зв'язку з ростом або розширенням бізнесу), це досить просте завдання: необхідно виміряти поточні показники продуктивності і ємності СЗД, після чого спланувати їхнє збільшення на найближчі рік-три й закупити відповідні компоненти СЗД.

Впровадження нового рішення теж, як правило, не викликає особливих труднощів: постачальники програмного забезпечення звичайно вже мають готові рекомендації з розгортання своїх систем. Приміром, у компанії VMware існує поняття «шаблонового користувача» стосовно до рішень по віртуалізації робітників місць (VDI). У прийнятій класифікації всі користувачі діляться на три категорії: light-користувачі несильно навантажують систему, тоді як medium- і heavy-користувачі більше вимогливі до ресурсів. По кожній категорії підготовлені кількісні характеристики: рекомендується емкость, що, пам'яті, число процесорних ядер, IOPS, обсяг переданих мегабайтів у секунду й так далі. Таким чином, знаючи, що необхідно розгорнути VDI-систему на 1000 користувачів, фахівці заздалегідь можуть оцінити, які IT-ресурси для цього будуть потрібні.

Однак бувають ситуації, коли інформація про вимоги до продуктивності дискової підсистеми відсутній або є обґрунтовані сумніви в застосовності шаблонових даних. У цьому випадку можна протестувати IT-систему на планованому до придбання встаткуванні. Багато виробників устаткування й програмних продуктів надають послугу, що дозволяє оцінити поводження конкретної IT-системи, що здійснює обробку певних даних, при різних програмно-апаратних конфігураціях і зрозуміти, що необхідно для її успішного впровадження.

Як показує досвід впровадження й експлуатації IT-систем, перед відновленням програмної або апаратної складової існуючої IT-системи необхідно зібрати статистику про продуктивність роботи всіх її компонентів – застосунків, серверів, СЗД – і повторити ту ж процедуру після впровадження. Це, по-перше, допоможе оцінити, наскільки виросла продуктивність кожного компонента IT-системи, а по-друге, дасть можливість продемонструвати бізнес-користувачам ефективність відновлення. Найчастіше оцінка досягнутих результатів залежить від суб'єктивної думки: в одного фахівця за консоллю встаткування працює повільніше, в іншого – швидше. Таким чином, маючи чіткі кількісні

характеристики, зафіксовані до й після відновлення, можна встановити, які зміни відбулися із системою насправді.

Самий оптимальний варіант використання дисків різних типів у СЗД – багаторівневе зберігання даних. По суті, цей не засіб збільшення продуктивності дискової підсистеми системи зберігання, а спосіб економії фінансів за рахунок установки в одному дисковому пулі декількох типів дисків, різних за рівнем продуктивності й вартості.

Яким образом досягається економія? В основу багаторівневого зберігання покладений наступний факт: дані, що зберігаються на СЗД, у більшості випадків затребувані нерівномірно. Так, у застосунку, що збирає й накопичує, скажемо, дані про погоду, активно використовуваних – або, у термінології багаторівневого зберігання, «гарячих» – не більше 5–15% від загального обсягу. Це пов'язане з тим, що свіжі дані запитуються набагато частіше, ніж, наприклад, дані п'ятирічної давнини.

Для зберігання «гарячих» даних завжди рекомендується використовувати високопродуктивні диски. Звертання до оставшихся «холодного» даним відбувається набагато рідше, і тому для їхнього зберігання краще задіяти більше дешеві і ємні, але менш продуктивні диски SAS або NL-SAS. При цьому система зберігання автоматично перерозподіляє активні й неактивні дані між відповідними швидкими й повільними рівнями зберігання.

Таким чином, ми одержуємо дисковий пул, продуктивність якого ледве менше продуктивності найшвидшого рівня зберігання в ньому, але загальна вартість дисків (з урахуванням ліцензії на багаторівневе зберігання для СЗД) виявляється істотно нижче, ніж якби в дисковому пулі використовувалися тільки високопродуктивні диски.

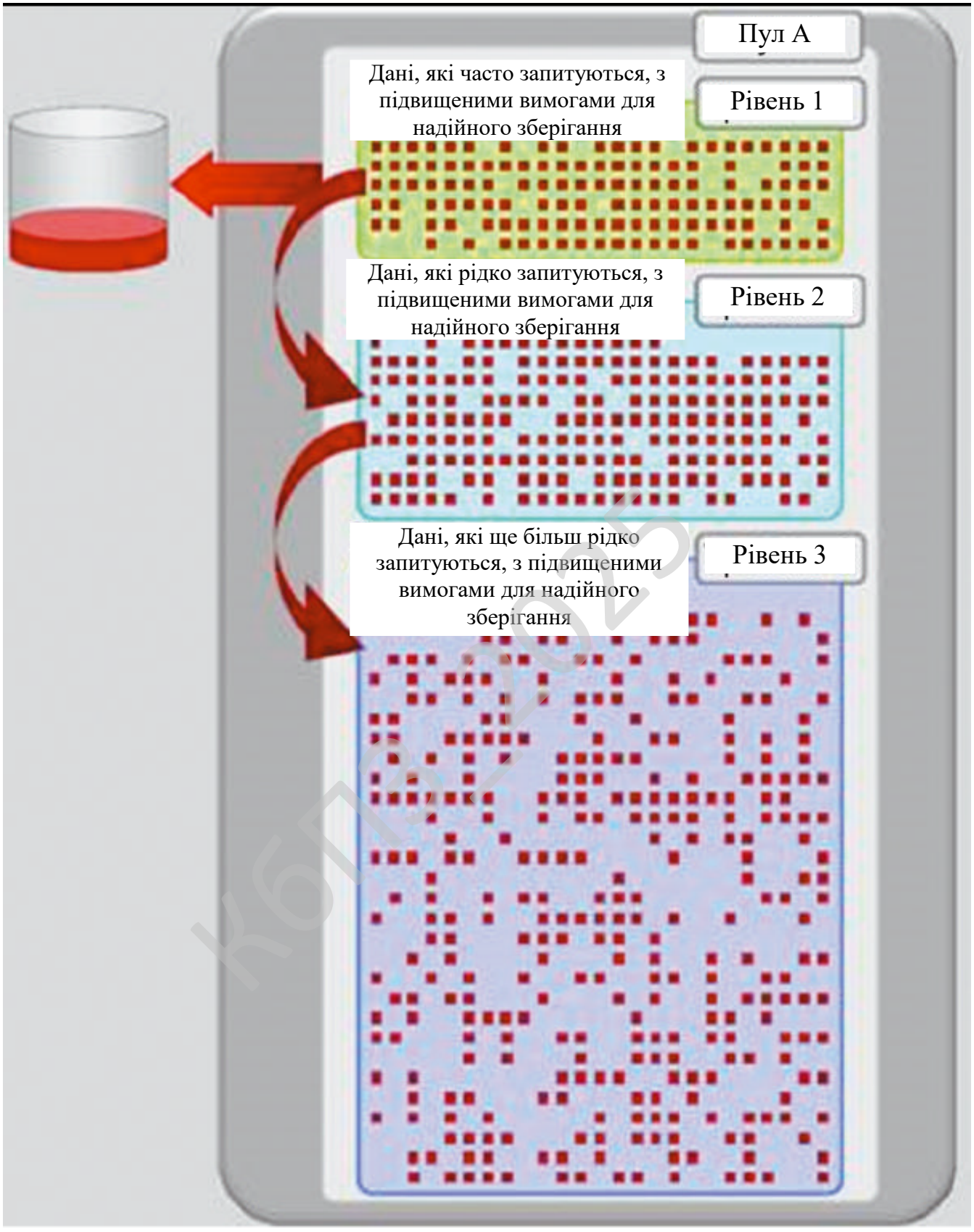

Рисунок 3.1 – Структурна схема системи

Основний критерій вибору багаторівневого зберігання, як уже було сказано, – нерівномірність використання даних. Найбільше часто багаторівневе зберігання використовується з базами даних OLTP і віртуальними середовищами, побудованими на основі VMware і Microsoft Hyper-V. Однак необхідно відзначити, що універсальних рецептів, що гарантують, що цей підхід буде ефективним абсолютно для всіх рішень, не існує. Якщо в компанії є сумніву в правильності застосування багаторівневого зберігання, вона завжди може протестувати свою IT-систему на конкретній системі зберігання даних і оцінити її ефективність.

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 оцінки продуктивності систем зберігання даних складається з наступних основних функціональних блоків:

– сам носій інформації – сервери СЗД;

– кодер Ріда-Соломона, який використовується, коли відбувається запис інформації на сервер СЗД, при цьому необхідно враховувати, що об’єм інформації, яка записується на диск серверу СЗД, 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.

Тобто реалізований проект призначена не тільки для оцінки продуктивності систем зберігання даних, але й для підвищення надійності зберігання даних.

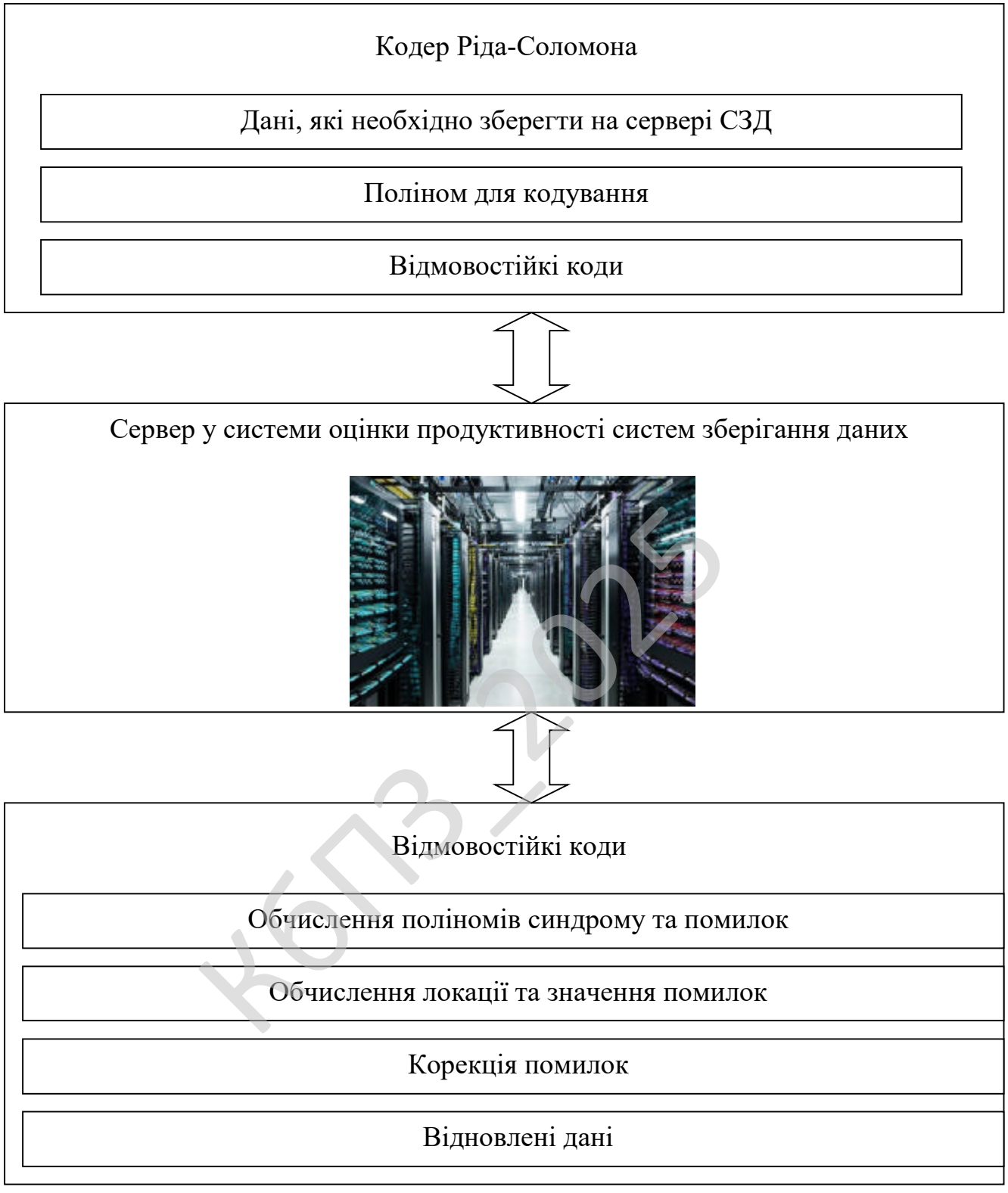

Рисунок 3.2 – Функціональна схема системи

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

– дані, які потрібно зберегти на сервері СЗД;

– поліном для кодування;

– відмовостійки коди.

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: $p(x) = i(x)\cdot x^{n-k} \bmod g(x)$.

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.

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

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

Теорема 3.1

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

Теорема 3.2

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

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

Матриця, що породжує

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

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

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

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

Матрицю *G* можна записати в символьній формі:

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

Перевірочна матриця

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

Тоді:

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

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

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

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

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

Властивості

Код Ріда-Соломона над $GF(q^m)$, що виправляє t помилок, вимагає $2t$ перевірочних символів і з його допомогою виправляються довільні пакети

довжиною t і менше. Відповідно до теореми про границю Рейгера, коди Ріда-Соломона є оптимальними з погляду співвідношення довжини пакета й можливості виправлення помилок – використовуючи $2t$ 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 «C» 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 $2t$, що багато менше ступеня кодового слова n. Для одержання відповідності між помилкою і її положенням у повідомленні будується поліном помилок.

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

Алгоритм Евкліда виконується наступним чином.

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

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

$$r_{n-1} = r_n q_n$$

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

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

Коректність цього алгоритму випливає з наступних двох тверджень:

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

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

Знаходження кореня

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

Визначення характеру помилки і її виправлення

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

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

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі. Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне

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

Діаграми потоків даних містять чотири типи елементів: Процеси які являють собою трансформацію даних в рамках описуваної системи; Сховища даних (репозиторії); Зовнішні по відношенню до системи сутності; Потоки даних між елементами трьох попередніх типів.

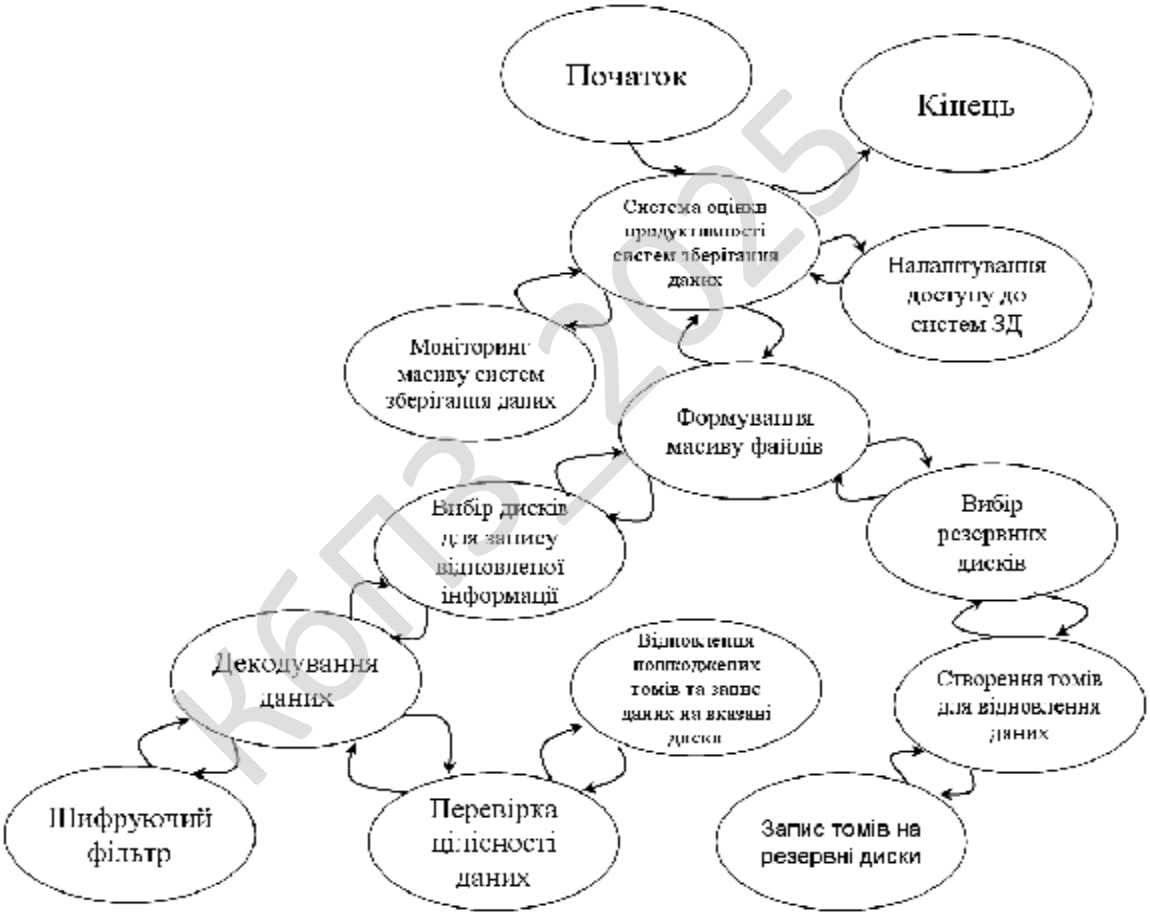

Рисунок 3.3 – Діаграма взаємодії процесів

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

Подійно-орієнтована архітектура (Event-driven architecture, надалі EDA) – шаблон архітектури програмного забезпечення, який призначений для створення подій, їх виявлення, споживання і реагування на них.

Подія може бути визначена як значна зміна стану. Наприклад, коли споживач купує автомобіль, стан автомобіля змінюється з "на продаж" до "продано". Архітектура системи дилера автомобілів може трактувати цю зміну стану як подію, поява якої може стати відомою іншим програмам даної архітектури.

З формальної точки зору, те, що виробляється, публікується, поширюється, виявляється і споживається (як правило, асинхронно) є повідомленням, яке називають сповіщенням про подію (або нотифікацією), а не самою подією, яка є зміною стану, що викликає появу повідомлення.

Події не подорожують, вони просто відбуваються. Проте термін подія часто використовується метонімічно для позначення самого нотифікаційного повідомлення, що може призвести до певної плутанини.

Цей архітектурний шаблон може застосовуватися при проектуванні і реалізації ПЗ і систем, які передають події між слабкозв’язаними компонентами програмного забезпечення і сервісами (службами).

Подійно-орієнтована система як правило складається з емітерів подій (або агентів) і споживачів подій (або стоків).

Стоки несуть відповідальність за здійснення реагування на появу події. Реакція не завжди може бути повністю забезпечена самим стоком. Наприклад,

стік, може бути відповідальним лише за фільтрацію, трансформацію і відправку події до іншого компонента або він може забезпечити повністю самостійну реакцію на таку подію. Перша категорія стоків може бути заснована на традиційних компонентах, таких як проміжне програмне забезпечення, орієнтоване на обробку повідомлень (message oriented middleware, MOM), в той час, як друга категорія стоків (самостійна реакція в режимі он-лайн) може вимагати більш придатної платформи (фреймворку) для виконання транзакцій.

Розробка ПЗ і систем в подійно-орієнтованій архітектурі дозволяє їм бути сконструйованими способом, який більш відповідає вимогам до їх створення, оскільки такі системи в більшій мірі пристосовуються до непередбачуваних і асинхронних середовищ.

Подійно-орієнтована архітектура (EDA) може доповнювати сервісно-орієнтовану архітектуру (SOA), оскільки сервіси (служби) можуть бути активовані тригерами, які ініціюються при настанні подій.

Ця парадигма особливо корисна, коли стік не забезпечує власного виконання будь-яких дій.

Подійно-орієнтована SOA (SOA-2) розвиває архітектури SOA і EDA для забезпечення більш глибокого і надійного рівня сервісів за рахунок використання раніше невідомих причинно-наслідкових зв'язків, щоб сформувати новий шаблон подій. Цей новий шаблон бізнес-аналітики дає поштовх до небаченого раніше зростання рівня автоматизації підприємства за рахунок привнесення додаткової цінної інформації в описану раніше модель діяльності.

Обчислювальна техніка та сенсорні пристрої (сенсори, датчики, контролери) можуть виявляти зміни стану об'єктів або умов і створювати події, які потім можуть бути оброблені сервісом (службою) або системою.

Подія може складатися з двох частин: заголовка події та тіла події. Заголовок події може включати в себе інформацію таку як, наприклад, назва події, часова мітка події і тип події. Тіло події – це частина, яка описує факт, що стався в

дійсності. Тіло події не слід плутати з шаблоном або логікою, яка може бути застосована як реакція на саму події.

Архітектура, керована подіями, складається з чотирьох логічних рівнів (шарів). Вона починається з виявлення факту, його технічного подання у формі події і закінчується непустою множиною реакцій на цю подію.

Генератор подій

Першим логічним шаром є генератор подій, який виявляє факт і представляє цей факт подією. Оскільки фактом може бути практично все, що може бути сприйнято, то ним може бути і генератор подій. Наприклад, генератором може бути клієнт електронної пошти, система електронної комерції або певний тип датчика.

Перетворення різних даних, отриманих від датчиків, в єдину стандартизовану форму даних, які можуть бути оцінені, є основною проблемою при розробці та реалізації цього шару. Однак, враховуючи, що подія є строго декларативною, можна легко застосовувати будь-які операції трансформації, тим самим усуваючи необхідність забезпечення високого рівня стандартизації.

Канал подій

Канал подій – це механізм, через який інформація від генератора подій передається до обробника подій (event engine) або стоку.

Це може бути з'єднання TCP/IP або вхідний файл будь-якого типу (простий текст, формат XML, e-mail тощо). В один і той же час може бути відкрито кілька каналів подій. Як правило, оскільки обробник подій повинен працювати в режимі, наближеному до реального часу, канали подій зчитуються асинхронно. Події зберігаються в черзі, очікуючи наступної обробки механізмом обробки подій.

Механізм обробки подій

Механізм обробки подій (event processing engine) є місцем, де подія ідентифікується і вибирається відповідна реакція на нього, яка потім виконується. Це також може призвести до породження ряду тверджень. Якщо подія, яка надійшла до механізму обробки подій, є наприклад такою «Запаси продукту ID

досягли нижнього допустимого рівня», це може ініціювати, наприклад, такі реакції як «Замовити продукт ID» і «Сповістити персонал».

Наступна подійно-орієнтована дія (післядія)

Щодо того, як можуть проявлятися наслідки події, слід відмітити, що вони можуть проявитись багатьма різними способами і у різноманітних формах (наприклад, повідомлення електронної пошти, надіслане комусь, або ПЗ, що виводить деяке попередження на екран). Залежно від рівня автоматизації, який забезпечується стоком (механізмом обробки подій), ці дії можуть виявитись зайвими.

Є три основні стилі обробки подій: простий, потоковий і складний. Часто ці три стилі використовуються спільно у розвинутій подійно-орієнтованій архітектурі.

Проста обробка подій

Проста обробка подій стосується подій, які безпосередньо належать до специфічних вимірних змін умов. У випадку простої обробки подій, мають справу з появою відомих подій, що ініціюють післядію (післядії). Проста обробка подій зазвичай використовується для управління потоком робіт в реальному часі, скорочуючи тим самим час затримки і вартість робіт.

Наприклад, прості події можуть створюватись (породжуватись) датчиком, що виявляє зміну тиску в шині або температуру навколишнього середовища.

Обробка потоку подій

При обробці потоку подій (event stream processing, далі ESP) відбуваються як звичайні, так і відомі події. Звичайні події (заявки, передачі RFID) перевіряються на те, чи є вони відомими, і передаються інформаційним передплатникам. Обробка потоку подій зазвичай використовується для управління потоком інформації в реальному часі і на рівні підприємства, що дозволяє своєчасно приймати рішення.

Обробка складних подій

Обробка складних подій (Complex event processing (CEP)) дозволяє за шаблонами простих і звичайних подій проводити аналіз того, чи наступила складна подія. Обробка складних подій полягає в оцінюванні взаємного впливу подій і в наступному виконанні дій. При цьому, типи подій (відомих або звичайних) можуть перетинатись, а події можуть виникати протягом тривалого періоду часу.

Кореляція подій може бути причинною, тимчасовою або просторовою. CEP вимагає використання складних інтерпретаторів подій, визначення і підбору шаблонів подій, а також відповідних кореляційних методів. Обробка складних подій зазвичай використовується для виявлення і реагування на аномальну поведінку, загрози і можливості у бізнесі.

Під час роботи над бакалаврською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

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

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

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

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

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

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

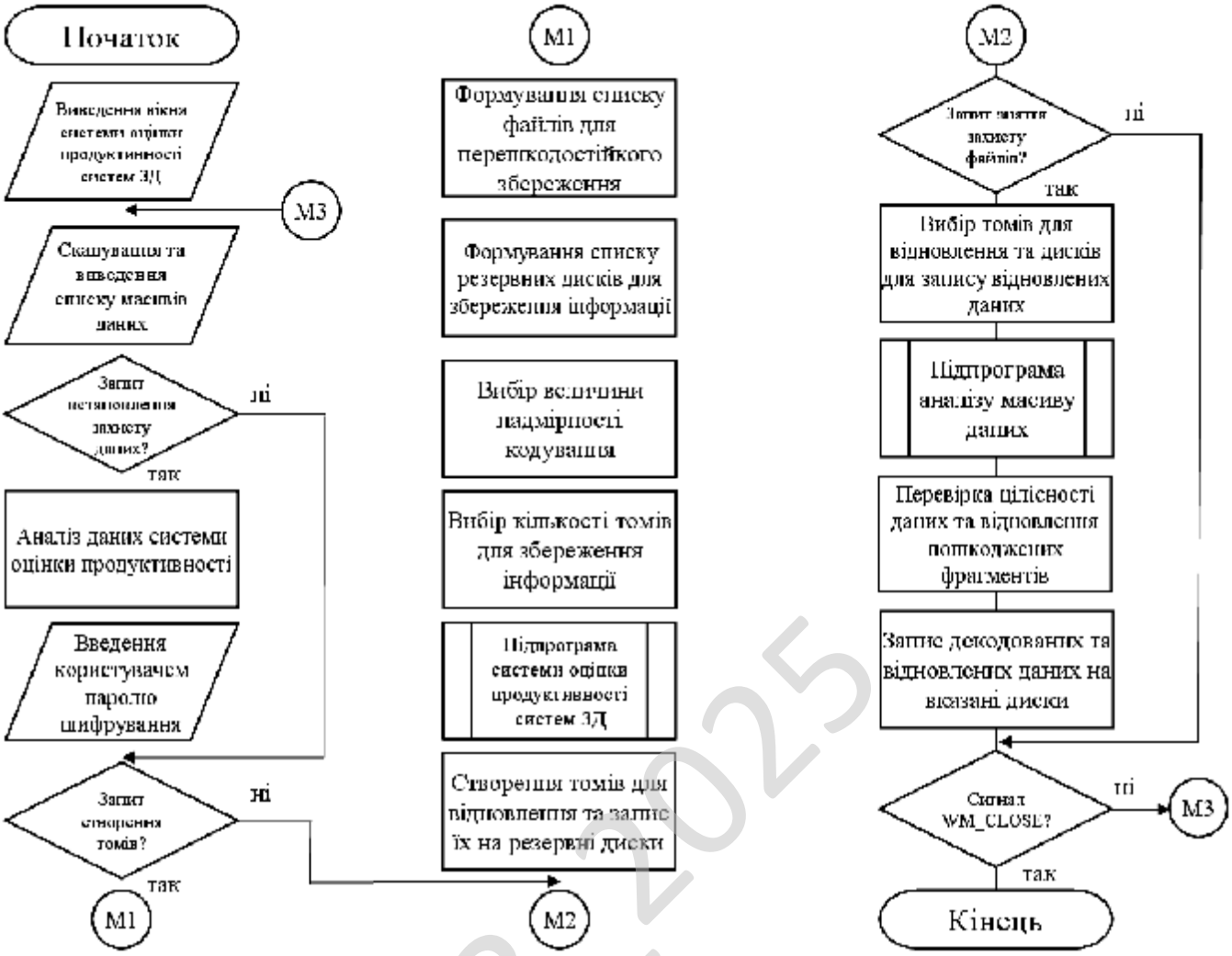

Рисунок 4.1 – Блок-схема основної програми

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

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

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

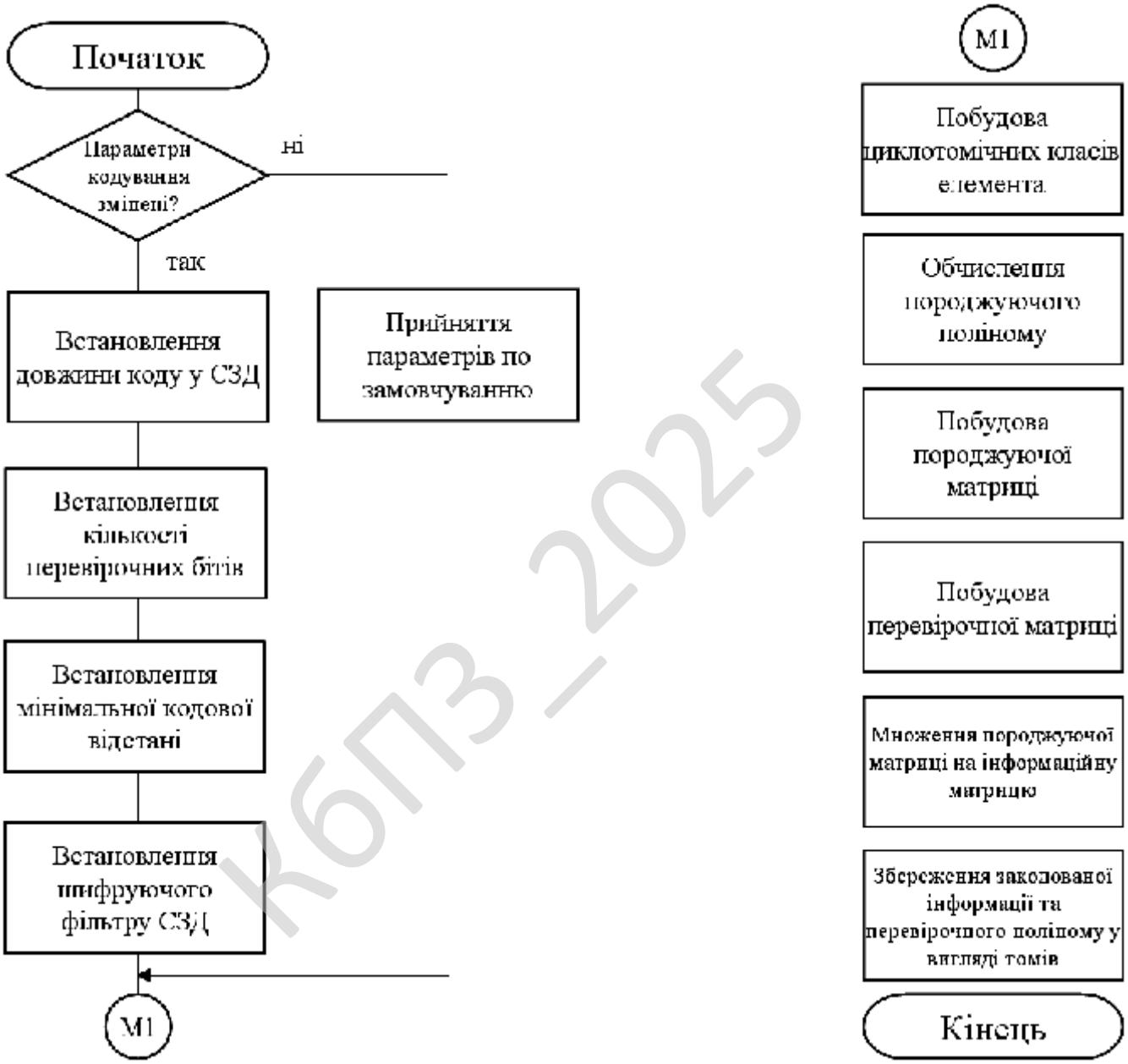

Рисунок 4.2 – Блок-схема роботи підпрограми

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще

більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

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

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

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

– Програмістам які реалізовують модулі інформаційної системи.

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

Під час роботи над програмою також використовувались підходи RUP.

Rational Unified Process(RUP) є ітеративним процесом розробки програмного забезпечення створеним Rational Software – підрозділом IBM з 2003.

RUP не є єдиним, конкретним розпорядчим процесом, а скоріше фреймворком процесу, що має бути адаптованим організаціями які займаються розробкою та командами розробників які оберуть елементи процесу, які підходять під їх потреби.

Rational Unified Process (RUP) являє собою продукт, спочатку розроблений Rational Software, яка була придбана компанією IBM в лютому 2003 року. Продукт містить у собі базу знань з гіперпосиланнями, та прикладами артефактів і

докладні описи для різних видів діяльності. RUP входить в продукт IBM Rational Method Composer (RMC), який дозволяє налаштування процесу.

До 1997 року, Rational придбав Verdix, Objectory, Requisite, SQA, Performance Awareness, та Pure–Atria. Поєднання баз досвіду цих компаній привело до вироблення семи «найкращих практик» сучасної програмної інженерії:

1. Розробляти ітеративно, керуючись ризиками.
2. Управляти вимогами.
3. Використовувати компонентну архітектуру.
4. Моделювати програмне забезпечення візуально.
5. Постійно перевіряти якість.
6. Контролювати зміни.
7. Підлаштовуватись.

Ці найкращі практики рухали розробку продуктів Rational, та використовувались польовими командами Rational, щоб допомогти клієнтам вдосконалити якість, та передбачуваність їх розробницьких зусиль. Щоб зробити ці знання доступнішими, Філіпу Крачтену, було поставлено завдання збирати явні фреймворки сучасної розробки програмного забезпечення. Ці зусилля використовував заснований на HTML механізм доставки процесів розроблений Objectory.

У результаті «Раціональний уніфікований процес» (RUP) завершив стратегічну опору для Rational:

– Адаптовний процес що направляє розробку

– Інструменти, що автоматизують використання цього процесу

– Сервіси, що прискорюють впровадження і процесу, і інструментів.

Будівельні блоки RUP

RUP заснований на наборі будівельних блоків, чи містить елементи, що описують те, що повинно бути зробленим, необхідні навички, та покрокове

пояснення того, як досягаються конкретні цілі розробки. Основними будівельними блоками, чи елементами вмісту, є наступні:

– Ролі (хто). Роль визначає набір навичок, компетенції та відповідальності.

– Робочі продукти (що). Робочий продукт являє собою щось отримане з завдання, в тому числі всі документи і моделі, випущені під час роботи впродовж процесу.

– Завдання (як). Завдання описує одиницю роботи, яка доручена ролі, яка забезпечує значущий результат.

У кожній ітерації, завдання діляться на дев'ять дисциплін: шість «інженерних дисциплін» (бізнес-моделювання, вимоги, аналіз і проектування, реалізація, тестування, розгортання) і трьох допоміжних дисциплін (конфігурація і керування змінами, управління проектами, середовища).

Чотири фази життєвого циклу проекту

RUP визначає життєвий цикл проекту, що складається з чотирьох фаз. Ці фази дозволяють процесу, бути представленим на високому рівні, подібно до того як представляються проекти у «водоспадному» стилі, хоча, по суті, ключем до процесу є ітерації розробки, які простягаються вздовж всіх фаз. Крім того, кожен етап має одну ключову ціль, та віху в кінці, яка позначає досягнення цілі.

Початкова фаза

Первинною ціллю є адекватна оцінка системи, як база для обчислення початкових розцінок та бюджету. На цьому етапі встановлюються бізнес випадки, які включають бізнес–контекст, фактори успіху (очікувані доходи, визнання на ринку, і т.д.), а також фінансовий прогноз. На додаток до бізнес випадку генерується базова модель прецедентів, план проекту, попередня оцінка ризику і опис проекту (основні вимоги до проекту, обмеження та основні характеристики).

Після їх завершення проект перевіряється на відповідність наступним критеріям:

– Зацікавленими сторонами досягають згоди з визначення масштабів і оцінкою вартості/термінів.

– Розуміння вимог як свідчення якості первинних прецедентів.

– Достовірність оцінок вартості/термінів, приорітетів, ризиків, та процесу розробки.

– Глибина і ширина будь-якого архітектурного прототипу, який був розроблений.

– Встановлення базової лінії за допомогою якої можна порівняти фактичні витрати в порівнянні із запланованим витратам.

Якщо проект не пройде цей етап, що називається віхою життєвого циклу, він може бути як скасований так і повторений після переконструювання з метою кращого задоволення критеріїв.

Фаза уточнення

Основна мета полягає в пом'якшенні ключових ризиків, виявлених на основі аналізу до кінця цієї фази. Фаза уточнення – фаза де проект починає набувати форми. На цьому етапі робиться аналіз предметної області, і архітектура проекту отримує свою базову форму.

Ця фаза має пройти віху життєвого циклу архітектури (LCA), задовольняючи такі критерії:

– Модель прецедентів, в якій ідентифікуються прецеденти та актори, та розробляється більшість описів прецедентів. Модель прецедентів повинна бути завершена на 80%.

– Опис архітектури програмного забезпечення в процесі розробки програмної системи.

– Виконувана архітектура, яка реалізує архітектурно значимі прецеденти.

– Бізнес це випадки та список ризиків що переглядаються.

– План розвитку проекту в цілому.

– Прототипи, що явно зменшили кожен виявлений технічний ризик.

Якщо проект не може переступити цю віху, ще є час для того, щоб він був скасований або змінений. Тим не менше, після закінчення цього етапу, проект

переходить в операцію з високим ступенем ризику, де зміни набагато складніші та згубні, при здійсненні.

Системна архітектура є ключовим елементом розробки, що отримується з аналізу предметної області.

Фаза конструювання

Основна мета полягає в створенні програмної системи. На цьому етапі основна увага приділяється розробці компонентів та інших характеристик системи. Це етап, коли відбувається основна частина кодування. У більш великих проектах, може бути кілька фаз конструювання, в спробі поділити прецеденти на керовані сегменти, які можуть утворити презентабельні прототипи.

Цей етап створює перший реліз програмного забезпечення. Його завершення позначає віха початкової боєготовності.

Фаза впровадження

Основна мета полягає в переведенні системи з розробки у продукт, зробивши її доступною та зрозумілою для кінцевого споживача. Діяльність у рамках цієї фази включає навчання кінцевих користувачів та обслуговуючого персоналу, бета–тестування системи для перевірки її на відповідність очікуванням користувачів. Продукт також перевіряється на відповідність рівню якості, встановленого в початковій фазі.

Якщо всі вимоги задоволені, досягається віха релізу продукту, і цикл розробки завершується.

Шість інженерних дисциплін

Дисципліни бізнес–моделювання

Бізнес–моделювання пояснює, як описати бачення організації, в якій буде розгортатись система і як використати це бачення для виділення процесу, ролей та обов'язків.

Організації стають все залежнішими від IT систем, що вимагає від інженерів інформаційних систем знання того, як застосунок що вони розробляють вписується в організацію. Підприємства інвестують в IT, коли вони розуміють,

конкурентні переваги і вартість що додає технологія. Метою бізнес–моделювання є по–перше встановити глибше розуміння та комунікаційний канал між бізнес інженерією та програмною інженерією. Розуміння бізнесу означає, що програмісти повинні розуміти структуру і динаміку цільової організації (клієнта), нинішні проблеми в організації, а також можливі удосконалення. Вони повинні також забезпечити загальне розуміння цільової організації між клієнтами, кінцевими користувачами та розробниками.

Дисципліни вимог

Вимоги пояснюють, як виявити запити зацікавлених осіб і перетворити їх в набір вимог, робочих продуктів, що осягають створювану систему й надають детальні вимоги до того, що система повинна робити.

Дисципліна аналізу та проектування

Метою аналізу і проектування, є показати, яким чином система буде реалізована. Ціллю є створення системи, яка:

– Виконує – в особливому середовищі реалізації – задачі та функції описані в описах прецедентів.

– Виконує всі свої вимоги.

– Легко змінити, коли змінюються функціональні вимоги.

Проектування дає в результаті модель проектування, а аналіз відповідно модель аналізу. Модель дизайну служить абстракцією вихідного коду; тобто модель дизайну працює «синькою», розміткою того як буде структурований та написаний вихідний код. Дизайн моделі складається проектування класів структурованих в пакети і підсистеми з чітко визначеними інтерфейсами, які представляють, що стане компонентами у реалізації. Він також містить опис того, як об'єкти цих сконструйованих класів співпрацюють для виконання прецедентів.

Дисципліна реалізації

Метою реалізації є:

– Визначити організацію коду з точки зору реалізації підсистем, які організовані в шари.

– Реалізація класів та об'єктів у термінах компонентів (вихідних файлів, виконуваних файлів, та інших).

– Для тестування розроблених компонент та модулів.

– Для інтеграції результатів, отриманих окремими виконавцями (чи групами) у виконувану систему.

Системи реалізуються через реалізацію компонентів. Процес описує як повторно використати існуючі компоненти, чи реалізувати нові компонетни з чітко визначеними відповідальностями, роблячи систему легше підтримуваною і збільшуючи можливості для повторного використання.

Дисципліна тестування

Цілі тестування:

– Перевірити взаємодії між об'єктами.

– Перевірити належну інтеграцію всіх компонентів програмного забезпечення.

– Щоб переконатися, що всі вимоги були правильно виконані.

– Щоб визначити та переконатись що дефекти будуть розглянуті до розгортання програмного забезпечення.

– Переконатись, що всі дефекти виправлені, повторно перевірені та закриті.

Раціональний уніфікований процес пропонує ітеративний підхід, а це означає, що тестування відбувається протягом всього проекту. Це дозволяє виявляти дефекти якомога раніше, що радикально знижує вартість виправлення дефекту. Тести проводяться за чотирма вимірами якості:надійності, функціональності, продуктивності додатків і продуктивності системи. Для кожного з цих вимірів критеріїв якості, процес описує як пройти життєвий цикл планування, проектування, виконання і оцінки тесту.

Дисципліна розгортання

Метою розгортання є успішно робити релізи продукту, та постачати програмне забезпечення для кінцевих користувачів. Вона охоплює широке коло

заходів, у тому числі виробництво зовнішніх версій програмного забезпечення, запаковування програмного забезпечення та бізнес–додатків, розповсюдження програмного забезпечення, встановлення програмного забезпечення та надання допомоги і підтримки для користувачів.

Хоча діяльність по впровадженню в основному зосереджена на перехідному етапі, багато які з цих заходів повинні бути включені в більш ранні етапи, щоб підготуватись до розгортання в кінці фази конструювання. Процеси Розгортування та середовища із RUP містять менше деталей, ніж інші робочі процеси.

Шість найкращих практик

Шість найкращих практик як описані в RUP є парадигмою програмної інженерії, яка перечислює шість ідей яким варто слідувати при конструюванні будь–якого проекту щоб мінімізувати провали, та збільшити продуктивність. Цими практиками є:

1. Ітеративна розробка. Найкраще було б знати всі вимоги наперед; тим не менш, часто це не той випадок. Існує кілька процесів розробки програмного забезпечення, які мають справу з рішеннями які дозволяють зменшувати вартість в термінах фаз розробки.

2. Управління вимогами. Завжди пам'ятати вимоги встановлені користувачами.

3. Використання компонент.

4. Розбиття складного проекту не тільки пропонується, а є фактично неминучим. Це дає можливість тестувати окремі компоненти до того, як вони будуть інтегровані в більшу систему. Також, повторне використання коду є великим плюсом та може бути здійснене легше через використання ООП.

5. Візуальне моделювання. Використовуйте діаграми щоб представити всі основні компоненти, користувачів, та їх взаємодії. «UML», скорочено від Unified Modeling Language, є інструментом що може зробити це завдання більш здійсненним.

6. Перевірка якості. Завжди робіть тестування більшої частини проекту в будь–який момент часу. Тестування стає важчим з розростанням проекту, та воно має бути постійним фактором в будь–якому створенні програмного продукту.

7. Контроль змін. Багато проектів створюються багатьма командами, іноді з різним місцезнаходженням, використовуючи різні платформи, і т.п. Як результат є важливим переконатись, що зміни які вносяться в систему синхронізуються та перевіряються постійно.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм REDOC III, який оперує з 80-бітовим блоком. Довжина ключа може змінюватися й досягати 2560 байт (204800 біт). Алгоритм складається тільки з операцій XOR над байтами ключа й відкритого тексту, перестановки й підстановки не використовуються.

1. Створюють таблицю ключів з 256 10-байтових ключів, використовуючи секретний ключ.

2. Створюють два 10-байтових блоки масок M1 і M2. M1 являє собою результат операції XOR перших 128 10-байтових ключів, а M2 – результат операції XOR других 128 10-байтовых ключів.

3. Для шифрування 10-байтового блоку:

a. Виконують операцію XOR з першим байтом блоку даних і першим байтом M1. Вибирають ключ у таблиці ключів, розрахованої в раунді 1. Використовують обчислене значення XOR як індекс таблиці. Виконують операцію XOR з кожним, крім першого, байтом блоку даних і відповідним байтом обраного ключа.

b. Виконують операцію XOR із другим байтом блоку даних і другим байтом M1. Вибирають ключ у таблиці ключів, розрахованої в раунді 1. Використовують обчислене значення XOR як індекс таблиці. Виконаєте операцію

XOR з кожним, крім другого, байтом блоку даних і відповідним байтом обраного ключа.

c. Продовжують ці дії з усім блоком даних (з 3-10 байтами), поки не буде використаний кожний байт для вибору ключа з таблиці після виконання операції XOR з ним і відповідним значенням M1. Потім виконують операцію XOR з кожним, крім використаного для вибору ключа, байтом, і ключем.

d. Повторюють етапи a-c для M2.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ системи оцінки продуктивності систем зберігання даних яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційне меню: Додатки; Налаштування системи оптимізації; Фільтри; Довідка.

– Розділу виведення результату роботи системи – файли.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

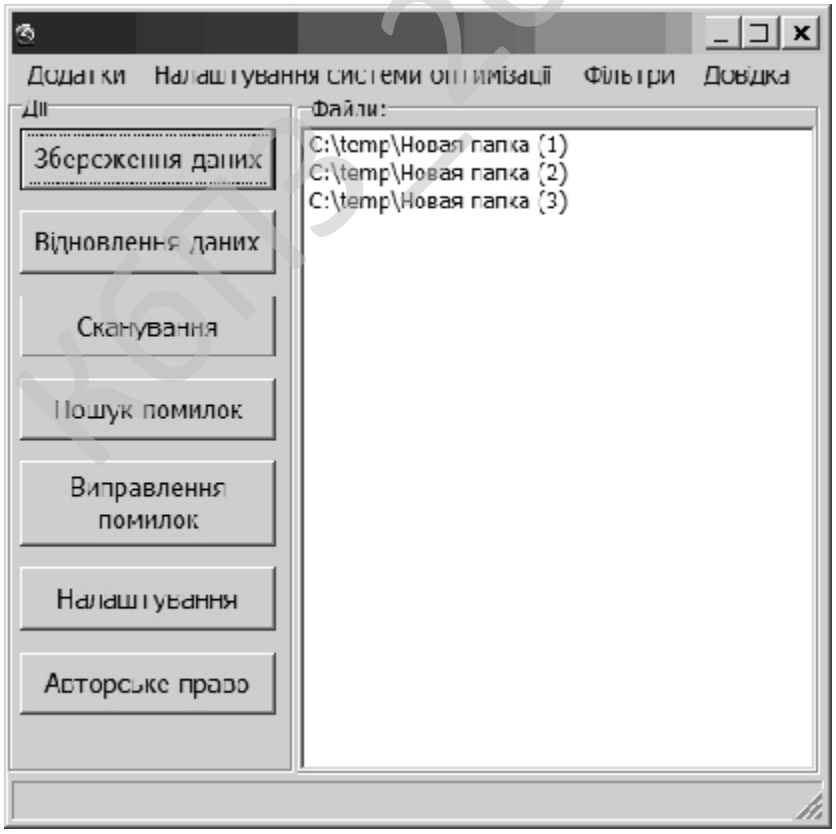

Рисунок 5.1 – Головне вікно ПЗ

Система призначена для оцінки продуктивності систем зберігання даних. У сучасних інформаційних інфраструктурах типу "хмар" або великих обчислювальних кластерів значний вплив на загальну продуктивність і якість роботи робить підсистема зберігання даних. На відміну від обчислювальної потужності й обсягів пам'яті, нарощуваних шляхом додавання серверів в інфраструктуру – сховище значно складніше піддається масштабуванню, що змушує особливо ретельно підходити до підбору його конфігурації.

Масування застосування технологій віртуалізації й широке поширення SSD-Носіїв створили одночасно й нові потреби й нові можливості в області побудови систем зберігання даних. Як наслідок – останнім часом з'явився ряд нових способів організації системи зберігання даних, у яких активно застосовуються нетривіальні підходи, такі як багаторівневе кешування, гетерогенні носії, віртуалізація системи зберігання. Їхнє поводження під різним навантаженням не завжди прогнозована й оцінка ефективності застосування того або іншого рішення вимагає розширеного тестування.

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

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

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

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

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

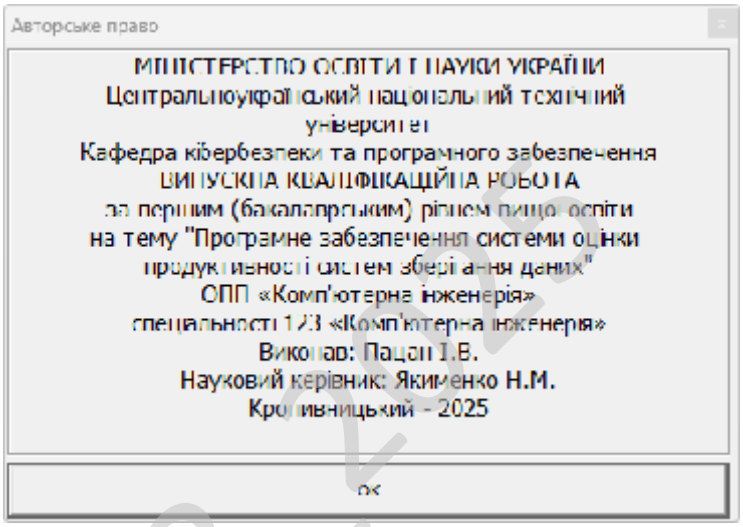

Рисунок 5.2 – Авторське право

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

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

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

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються в IT рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

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

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

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

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

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

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

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

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

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

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

– У програмі можуть бути пропущені деякі маршрути.

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

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

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

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

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

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

Проводилось тестування чорної скриньки.

Основне місце програми тестів «чорної скриньки» — інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

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

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

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

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

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

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

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITс.

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

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

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій.

– Помилок інтерфейсу.

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

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

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Shareware.

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

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

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час

завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

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

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

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи оцінки продуктивності систем зберігання даних.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем оцінки продуктивності систем зберігання даних.

– Досліджена система оцінки продуктивності систем зберігання даних.

– На основі отриманих результатів досліджень створена програмна реалізація системи оцінки продуктивності систем зберігання даних.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання оцінки продуктивності систем зберігання даних.

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

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

Програма реалізована на мові високого рівня Visual C#. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи оцінки продуктивності систем зберігання даних. 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 REDOC III.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Scott Jernigan «CompTIA Network+ Certification All-in-One Exam Guide, Eighth Edition». 2022. – 976 p.

2. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.

3. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

4. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

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

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

7. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

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

9. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

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

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

12. 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.

13. 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.

14. 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.

15. 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.

16. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379.

17. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated

with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645.

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

19. 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.

20. 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.

21. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

29. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

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

31. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

32. 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.

33. 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.

34. 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.

35. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

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

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

38. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в

комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

39. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

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

41. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

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

43. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139

44. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

45. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу

відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

46. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

47. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.

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

49. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Алгоритми формування безлічі маршрутів передачі метаданих у антивірусні хмарні системи. Збірник наукових праць "Системи обробки інформації". - Випуск 5 (142). - Х.: ХУПС - 2016. - С. 148-152.

50. Смірнов О.А., Смірнов С.А. Дідик А.К., Дрєєв О.М. Моделі системи нейромережевих експертів безпечної маршрутизації у хмарних антивірусних системах. Збірник наукових праць "Системи обробки інформації". - Випуск 3 (140). - Х.: ХУПС - 2016. - С. 36-39.

51. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Спосіб контролю ліній зв'язку телекомунікаційної системи антивірусу. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. - 2016. - С. 121-127.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи оцінки продуктивності систем зберігання даних.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 48-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи оцінки продуктивності систем зберігання даних.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи оцінки продуктивності систем зберігання даних;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Visual C#.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 102 аркуші.

8 Етапи розробки

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

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

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

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

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

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Якименко Н.М.

Програмне забезпечення системи оцінки продуктивності систем зберігання даних

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 56

Літера: РП

Кропивницький – 2025 року

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[]
{
            "БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА",
            "",
            "На тему:",
            "",
            "Програмне забезпечення системи оцінки продуктивності систем
зберігання даних ",
            "",
            "",
            "Керівник: Якименко Н.М.",
            "",
            "Розробив: студент Пацан Ігор Вікторович",
            "                  гр. КІ-22-мб",
            "",
            "м. Кропивницький 2025"});
            this.developersListBoxdevelopersListBox.Location = new
System.Drawing.Point(11, 6);
```

```
            this.developersListBoxdevelopersListBox.Name =
"developersListBoxdevelopersListBox";
            this.developersListBoxdevelopersListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.developersListBoxdevelopersListBox.Size = new
System.Drawing.Size(330, 182);
            this.developersListBoxdevelopersListBox.TabIndex = 0;
            this.developersListBoxdevelopersListBox.TabStop = false;
            this.developersListBoxdevelopersListBox.SelectedIndexChanged += new
System.EventHandler(this.developersListBoxdevelopersListBox_SelectedIndexChanged
);
            //
            // 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;

    }
}
```

Файл Code_Rid_Solomon_Encoder.cs - кодер коду Ріда-Соломона для системи оцінки продуктивності систем зберігання даних

```
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 мережних системах зберігання

```
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
звороткої
                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 і-ой рядка
                    int i_n = i * this.n;

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

                        int idx = i_n + j;

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

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

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

```
                    int i_n = (i * this.n);

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

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

                // Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((k % progressMod1) == 0)
                     &&
                        (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
    }
}
```