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

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації”

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

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Якименко Н.М.
«____» _____ 2024 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Центр *Заочної та дистанційної освіти*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *магістр*
Галузь знань _12 _____*"Інформаційні технології"*_____
Спеціальність 123 *"Комп'ютерна інженерія"*
Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерна інженерія"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2024 року

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

Віднічук Ганні Миколаївні
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації*

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 № 21-13 від 07.08.2024 року

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

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації*

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

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

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

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

6. Консультанти розділів роботи

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

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

КАЛЕНДАРНИЙ ПЛАН

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

Дата видачі завдання
« 6 » вересня 2024 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2024 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Віднічук Г.М. Дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи безперервної роботи додатків при серверній віртуалізації.

Метою розробки є дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації.

Об'єктом дослідження є процес безперервної роботи додатків при серверній віртуалізації.

Предметом дослідження є методи безперервної роботи додатків при серверній віртуалізації.

Методи дослідження базуються на методах теорії комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи безперервної роботи додатків при серверній віртуалізації.

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, серверна віртуалізація

ABSTRACT

Vidnichuk H.M. Research and software implementation of a system of continuous operation of applications in server virtualization. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for the system of continuous operation of applications during server virtualization.

The purpose of the development is research and software implementation of a system of continuous operation of applications in server virtualization.

The object of the study is the process of continuous operation of applications during server virtualization.

The subject of research is methods of continuous operation of applications in server virtualization.

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

The result of the work is the software implementation of the system of continuous operation of applications during server virtualization.

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

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

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

The program was developed in the Visual C# environment.

Keywords: computer engineering, server virtualization

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ВМ	–	Віртуальна машина
ОС	–	Операційна система
ПЗ	–	Програмне забезпечення
СЗД	–	Система зберігання даних
ЦОД	–	Центр обробки даних
BIOS	–	Basic Input-Output System
DRaaS	–	Сервіс відновлення як послуга
LINQ	–	Language Integrated Query
RAID	–	Redundant Array of Inexpensive Disks

ВСТУП

Актуальність теми. Широке поширення серверної віртуалізації й хмарних технологій вимагає нового підходу до реалізації рішень, що забезпечують безперервну роботу бізнес-додатків у випадку великих аварій і катастроф.

Виробники серверів і систем зберігання постійно вдосконалюють використовувані в цих продуктах засобу для запобігання апаратних і програмних збоїв, їхньої своєчасної ідентифікації й діагностики, а також оперативного усунення таких збоїв. У цьому зв'язку можна згадати спеціальні сервісні процесори, що контролюють стан устаткування, дублювання основних компонентів системи, функції гарячої заміни жорстких дисків, вентиляторів і блоків живлення. Нарешті, об'єднання двох і більше серверів в відказостійкий кластер дозволяє забезпечити продовження роботи додатків навіть у випадку повного виходу з ладу одного з них.

Однак всі ці міри виявляються неефективними, якщо з якихось причин буде зупинений весь центр обробки даних – наприклад, через повінь, пожежі або багатогодинного відключення електрики. Оскільки великі природні й техногенні катастрофи звичайно охоплюють цілі міста або регіони, захистити від них критичні для бізнесу додатка можна одним способом: побудувати віддалений на десятки або сотні кілометрів резервний ЦОД, щоб оперативно запустити там додатка у випадку подібних ситуацій. В ідеалі й той і інший повинні бути оснащені однаковими серверами й системами зберігання даних.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації.

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

– Огляд існуючих систем безперервної роботи додатків при серверній віртуалізації.

– Дослідження системи безперервної роботи додатків при серверній віртуалізації.

– Програмна реалізація системи безперервної роботи додатків при серверній віртуалізації.

Об'єктом дослідження є процес безперервної роботи додатків при серверній віртуалізації.

Предметом дослідження є методи безперервної роботи додатків при серверній віртуалізації.

Методи дослідження базуються на методах теорії комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод безперервної роботи додатків при серверній віртуалізації.

– Розроблено вітчизняний продукт безперервної роботи додатків при серверній віртуалізації, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі безперервної роботи додатків при серверній віртуалізації.

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

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

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Розроблюване в даній роботі програмне забезпечення дозволяє включити відновлення як послугу (DRaaS) у комплексну стратегію забезпечення доступності даних. Ви зможете ефективніше використовувати вкладення у віртуальне середовище й системи зберігання, а також підключити гібридна хмара до наявної інфраструктури.

З розроблювальним програмним забезпеченням реплікація ВМ на рівні образа не вимагає більших витрат. Вона забезпечить можливість післяаварійного відновлення із хмари для всіх додатків (RTPO менш 15 хвилин) і включає кращих провайдерів DRaaS по усьому світі.

Сьогоднішні проблеми DRaaS

DRaaS викликає великий інтерес, але користуватися цією послугою непросто (по відкликаннях замовників). Багато продуктів сьогодні пропонують DRaaS як окрему послугу або як доповнення до вже існуючої стратегії захисту даних. Найчастіше це означає додаткові вкладення, підвищення навантаження для IT-фахівців і ускладнення існуючої інфраструктури. При цьому немає гарантій відновлення даних у випадку аварії.

Більшість рішень для DRaaS пропонують досить обмежені можливості. У недавньому звіті провідної аналітичної компанії повідомляються: замовники в цілому задоволені якістю послуг, але IT-підрозділи намагаються знайти спосіб зробити ці послуги рентабельними для компанії.

– Деякі DR-системи використовують бекапи замість реплік, причому послуга однаково називається DRaaS. Використання бекапів для DRaaS не завжди ефективно, а показники RTO і RPO далекі від тих значень, які потрібні для сьогодення післяаварійного відновлення.

– Деякі рішення DRaaS дозволяють виконувати реплікацію, але для використання інших можливостей користувачам і сервіс-провайдерам доводиться здобувати додаткові продукти. Це стосується навіть таких важливих областей, як бекап і робота з мережею. Вартість таких рішень не відповідає пропонованим можливостям.

– Деякі постачальники DRaaS-рішень конкурують із сервіс-провайдерами. Вони одночасно рекламують власні дата-центри й намагаються продавати свої рішення провайдерам.

Ми розглядаємо DRaaS у рамках комплексної стратегії доступності даних. Наше рішення поєднує можливості бекапа й реплікації, підтримує роботу із хмарою й пропонується за доступною ціною. З розробленим програмним забезпечення одержуємо:

– Ефективну реплікацію на рівні образа для сьогодення DR – RTPO мінімальні.

– Просту модель ліцензування й обґрунтовану ціну – для користувачів, оренду по кількості ВМ – для сервіс-провайдерів.

– DR для будь-яких додатків – ефективно, не вимагає більших витрат і підходить для будь-яких СЗД, додатків і ОС.

– Розширені можливості роботи з мережею – не вимагають складної конфігурації або використання VPN, що звичайно представляє основну проблему при тестуванні DR і аварійному перемиканні.

Докладніше про розширені можливості роботи з мережею для DRaaS:

– Якщо ви пропонуєте послуги DR, мережа може стати однією з основних проблем. На відміну від інших рішень, програмне забезпечення, що розробляється, дозволяє управляти не тільки реплікацією даних для DR, але й мережними підключеннями між двома площадками, незалежно від їхнього фізичного місця розташування. При тестуванні DR або аварійному перемиканні обмін даними між ВМ підтримується автоматично, що спрощує роботу з мережею, знижує витрати й дозволяє обійтися без зміни мережних налаштувань. Вам не прийде здобувати додаткове ПЗ або реплікувати налаштування TCP/IP до, після або під час післяаварійного перемикання.

Ефективне й доступне катастрофостійке рішення (DR) у хмарі:

– Для провайдерів програмне забезпечення, що розробляється, – простий спосіб надати послугу DRaaS, а для замовників – зручний спосіб її використовувати. IT-підрозділи зможуть скористатися перевагами гібридної хмари й при цьому підтримувати прямий контакт із сервіс-провайдером. Таким чином, замовники можуть перекласти частина відповідальності на довіреного сервіс-провайдеру. При цьому можна вибирати тільки дійсно необхідні послуги й самостійно контролювати завдання реплікації.

Завдяки реплікації ВМ на рівні образа програмне забезпечення, що розробляється, забезпечує швидке й надійне післяаварійне відновлення через провайдеру.

1.2 Область застосування

Програмне забезпечення, що розробляється, для сервіс-провайдерів спрощує надання послуг DRaaS, робить цей бізнес більше вигідним, а послуги – привабливими для будь-якого замовника. Переваги для провайдерів:

– Незалежність від стороннього ПЗ – буде потрібно тільки гіпервізор. Не потрібно здобувати додаткове ПЗ для керування аварійним перемиканням, оптимізації роботи з мережею або WAN-акселерації. Це знижує сукупну вартість володіння (TCO), підвищує ваш дохід і робить послугу більше вигідної для замовників.

– Просте ліцензування на основі щомісячної оренди по кількості ВМ. Обмежень на мінімальну кількість ВМ немає.

– Готовий ринок і канали поширення.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

У цьому розділі коротко описані основні методи забезпечення катастрофостійкості, які дозволять відновити працездатність IT-сервісів із заданим показником актуальності даних (RPO, Recovery Point Objective, точка відновлення – максимальний припустимий період часу, за який дані можуть бути загублені) і в заданий термін (RTO, Recovery Time Objective, час відновлення – максимальний припустимий час, у плині якого дані повинні бути відновлені) при повнім або частковому припиненні роботи цих сервісів на основній площадці. Як резервна площадка може виступати продуктивний резервний ЦОД, виділені потужності на площадці сервіс-провайдеру або потужності, надавані по певному SLA, в «хмарі» або в публічному ЦОД. Природно, чим вище цільові показники RPO і RTO, тим вище вимоги до резервних площадок, каналам зв'язку між ними й до методів забезпечення катастрофостійкості, а як наслідок – і вартість рішення. Тому при плануванні таких рішень потрібно правильно класифікувати сервіси, урахувати зв'язки між ними, визначити для них реальні показники RPO і RTO і потім підібрати метод забезпечення катастрофостійкості, здатний їх забезпечити. Отже, розглянемо доступні методи в порядку зростання вартості їхньої реалізації.

Резервне копіювання

Самий недорогий метод забезпечення катастрофостійкості, не потребуючий розвитий інфраструктури ні на основній, ні на віддаленій площадках, що забезпечує цілісність даних на певний момент часу, з низькими показниками RPO і RTO. Розглянемо основні його різновиди.

На стрічки

Самим доступним способом забезпечення катастрофостійкості є резервне копіювання на стрічки. З певною періодичністю стрічки з актуальними резервними копіями виносяться на іншу площадку, у банк, у сейф і т.д. У випадку катастрофи дані з останньої копії послідовно відновлюються на існуючі в резерві або знову куплені/орендовані обчислювальні потужності. Важливо помітити, що разом зі стрічками бажано зберігати й стример, що зміг би прочитати стрічки відповідного стандарту.

Такий підхід, як правило, забезпечує не кращі показники RPO і RTO, а також вимагає додаткових витрат на логістику й зберігання, відрізняється невисоким показником надійності. Час відновлення даних може становити до двох доби для першого сервісу, приміром, якщо аварія відбулася в суботу, а стрічки перебувають у банку.

Відновлення резервних копій відбувається послідовно й, приміром, на відтворення інфраструктури з 30 серверів (віртуальних і фізичних) може знадобитися кілька доби, і це у випадку наявності необхідних обчислювальних ресурсів. RPO залежить від частоти вивозу стрічок і в середньому становить один тиждень (якщо стрічки вивозити за межі ЦОД, приміром, по суботах після повного резервного копіювання інфраструктури).

Якщо мова йде про декілька продуктивних ЦОД або серверних, то витрати на доставку стрічок у місце їхнього зберігання можуть наближати це рішення за вартістю до наступного методу резервного копіювання – на диски.

Плюси й мінуси:

«+» низька вартість. Резервне копіювання є невід'ємної складової IT в організації й на сьогоднішній день найчастіше виконується на стрічки. У такому випадку витрати для забезпечення катастрофостійкості полягає тільки в логістиці й зберіганні стрічок;

«+» більшість приводів для читання/запису стрічок у базі надає функцію шифрування, що може зіграти ключову роль при виборі недорогого рішення;

«–» гірші показники часу й точки відновлення: процес відновлення виконується послідовно, для пошуку потрібної резервної копії необхідно перемотати одну або кілька стрічок у потрібне місце, а вуж потім відновлювати дані; як правило вивіз стрічок за межі ЦОД виконується не частіше рази в на добу, а в середньому – раз у тиждень;

«–» низька надійність зберігання на стрічках. Є ймовірність ушкодження даних на стрічці під час перевезення й зберіганні;

«–» людський фактор. Велика кількість ручної роботи як у випадку аварії, так і при обслуговуванні підвищує ймовірність помилок і порушення SLA.

На диски

Резервне копіювання на диски (D2D, Disk-to-Disk) для забезпечення катастрофостійкості, має сенс для компаній з декількома серверними або ЦОД. Копії, зроблені локально, передаються на одну або кілька віддалених площадок по каналах зв'язку. У випадку втрати даних або краху однієї із площадок, резервні копії передаються на вихідну площадку або відновлюються на інший, послу чого сервіс надається віддалено.

Сучасні рішення резервного копіювання на диски, як правило, у базі надають функції дедуплікації, реплікації, передачі даних по тонких каналах зв'язку, що спрощує дії адміністратора, знижує вимоги до комунікацій і зменшує обсяг збережених даних на дорогих, порівняно зі стрічками, носіях.

У першу чергу D2D скорочує час, необхідне для резервного копіювання й відновлення даних, тому що відпадає необхідність у перемотуванні стрічок, а функції дедуплікації й реплікації дозволяють оптимізувати RPO. Для самих критичних додатків цей показник може досягати 15 хв при використанні інкрементальних копій, і до доби для самих об'ємних і/або менш критичних сервісів.

Показники RTO у такому сценарії набагато краще, ніж у випадку зі стрічками, тому що резервні копії перебувають безпосередньо в ЦОД, їхній пошук і відновлення займають менше часу, можна відновлювати кілька додатків

одночасно. Таким чином, при наявності заздалегідь підготовлених обчислювальних потужностей, перший сервіс можна відновити менш чим за годину, а інфраструктуру з 30 серверів – скажемо, за добу.

Плюси й мінуси:

«+» досить бюджетне рішення, що комбінує необхідне для кожної компанії резервне копіювання із забезпеченням катастрофостійкості. Не вимагає обов'язкової наявності всіх обчислювальних ресурсів на віддаленій площадці;

«+» висока гнучкість при виконанні резервних копій, що дозволяє регулювати точку відновлення даних;

«+» можливість зберігання декількох копій даних у різних територіально-розподілених місцях;

«–» залежність відновлення даних на віддалену площадку від каналів зв'язку, що в українських реаліях може мати істотне значення;

«–» людський фактор, внаслідок великої кількості ручної роботи, що підвищує ймовірність помилок і порушення SLA.

Засобами додатка/сервісу

Деякі додатки, сервіси, СУБД мають власні механізми забезпечення катастрофостійкості, які дозволяють досягти найкращих показників RPO і RTO, але, найчастіше, вимагають на резервній площадці таких же потужних серверів і СЗД як і на основний, а також висувають високі вимоги до каналів зв'язку й організації мережі. До таким відносяться SQL Server, Oracle Database, Exchange, сервіси AD, DNS та ін. Як приклад більш докладно розглянемо Microsoft SQL Server.

Актуальна версія SQL Server 2012 пропонує наступні можливості по забезпеченню високої доступності, відказостійкости й катастрофостійкості:

Катастрофостійке рішення засобами SQL Server 2012 можливо тільки з використанням опції AlwaysOn Availability Group. У синхронному режимі досягаються кращі показники RPO і RTO, але з падінням продуктивності. Аргументом на користь асинхронного режиму (а також опції AlwaysOn Failover

Cluster Instance) може послужити швидкість роботи СУБД. Справа в тому, що при використанні механізмів синхронної реплікації балок AlwaysOn Availability Group зростає час на відпрацьовування транзакції за рахунок латентності мережі – транзакція не буде завершена доти, поки дані не будуть записані на віддалену площадку.

Для забезпечення комплексного захисту даних можливо спільне використання декількох зазначених опцій. На рисунку нижче AlwaysOn Failover Cluster Instance забезпечує відказостійкість у межах однієї площадки, а AlwaysOn Availability Group в асинхронному режимі – катастрофостійкость на двох площадках.

Проте, нерідкі випадки, коли для сервісів з убудованими засобами реплікації все-таки вибираються інші методи забезпечення катастрофостійкості. Основними причинами є складність і/або дорожнеча реалізації, які можуть бути зв'язані з вартістю встаткування, каналів зв'язку, необхідних ліцензій.

Плюси й мінуси:

«+» найкращі показники RTO і RPO;

«+» гарантія цілісності даних;

«+» автоматичний або напівавтоматичний процес відновлення працездатності сервісу, що знижує вплив людського фактора;

«–» вартість рішення, обумовлена необхідністю дублювати встаткування;

«–» підтримується в порівняно невеликій кількості додатків.

Засобами СЗД

Цей спосіб має на увазі реплікацію даних системами зберігання на блоковому рівні між декількома площадками. При цьому СЗД байдуже, які сервери до неї підключені, які додатки на них виконуються та ін., вони лише гарантують, що кожний блок даних, записаний на одній, буде в певний строк продубльований на іншій.

При цьому виділяють наступні стандартні типи реплікації: синхронна, асинхронна, асинхронна Remote Snap. Також у кожного виробника СЗД є

пропрієтарні технології, які декілька відрізняються від стандартних, але здебільшого на них базуються. Приміром, архітектура СЗД HP StoreVirtual 4000 (раніше LeftHand), які орієнтовані на віртуалізовані середовища й територіально рознесені інфраструктури, дозволяє записувати й читати дані на обох площадках одночасно (що не застосовно в описані нижче випадках). Таким чином, реплікуємі дані не блокуються на резервній площадці, а залишаються доступними й на читання, і на запис, що дозволяє будувати розподілені кластери.

Синхронна реплікація

Гарантує, що блоки даних, записані на основну СЗД, будуть репліковані на віддалену без усяких затримок. По суті, основна чекає від віддаленої підтвердження запису блоку, а лише потім сама видає підтвердження додатку. У такий спосіб RPO у цьому випадку прагне до нуля. Приміром, якщо СУБД завершила транзакцію й після цього відбувся збій, те відновлена база даних на іншій площадці буде в актуальному стані з точністю до останньої завершеної транзакції.

Цей тип реплікації характеризується високими вимогами й обмеженнями, які можуть декілька мінятися залежно від виробника СЗД. Звичайна відстань між реплікуємими СЗД повинне бути в межах 40 км, а латентність каналу зв'язку не повинна перевищувати 2 мс (темна оптика) – при цьому додаток повинне допускати вдвічі більші затримки.

Асинхронна реплікація

Відрізняється від синхронної тем, що СЗД спочатку локально записує дані й відправляє підтвердження додатку, а паралельно передає дані на другу СЗД. Таким чином, у порівнянні із синхронною реплікацією, знімаються обмеження по відстані й латентності каналу зв'язку, але при цьому збільшується значення RPO.

Асинхронна реплікація Remote Snap

Має на увазі використання знімків. Основна СЗД обмінюється знімками томів з резервної СЗД безупинно через задані інтервали часу. Це дозволяє

відновити дані з резервної площадки на момент, коли був зроблений (і успішно переданий) останній знімок основний СЗД.

Звичайно виробники рекомендують інтервал реплікації не менш 15 хв, у такий спосіб краще значення RPO становить 15 хв плюс час, необхідне для передачі цього знімка на другу площадку.

Організація

Як правило, виділяють продуктивну площадку, на якій сервіси працюють в активному режимі, і резервну, на якій розташовується СЗД і серверна інфраструктура, що повністю дублює основну або достатня для відновлення самих критичних сервісів. У такому випадку використовується однобічна реплікації з активної площадки на резервну. З метою економії засобів, іноді площадки використовуються в режимі Active-Active, коли обидві є продуктивними й мають якийсь запас серверної потужності для відновлення сервісів один одного. У такому випадку має місце двостороння реплікація. Всі перераховані вище методи реплікації мають на увазі, що основний логічний диск доступний на читання й запис, а реплікуємий тільки на читання або не доступний взагалі до завершення реплікації.

У випадку катастрофи на основній площадці, необхідно запустити до виконання заздалегідь прописаний Disaster Recovery Plan, що має на увазі послідовність дій для реанімації сервісів на резервній площадці. При цьому існує ряд програмних продуктів, здатних звести необхідні дії адміністратора до натискання декількох кнопок, що, у свою чергу, у рази знижує значення RTO.

Плюси й мінуси:

«+» найкращі RPO і RTO для всіх сервісів;

«+» ПЗ автоматизації процесу відновлення мінімізує можливі помилки;

«+» інтеграція із самими популярними гіпервізорами;

«–» висока вартість, обумовлена дублюванням устаткування на двох площадках, а також придбанням ПЗ автоматизації процедур перемикання активної площадки на резервну;

«–» високі вимоги до каналів зв'язку, особливо при синхронній реплікації.

Висновки

Розглянуті методи забезпечення катастрофостійкості найчастіше використовуються спільно в межах однієї компанії для забезпечення необхідних показників RPO і RTO для різних груп сервісів. Реалізація таких рішень – непростий процес, що вимагає скрупульозного підходу на всіх етапах, починаючи з визначення вимог по кожному сервісі й закінчуючи процедурою впровадження й тестування. Треба розуміти, що чим жорсткіше висунуті до рішення вимоги, тим дорожче воно обійдеться. Найчастіше трапляється так, що при першій спробі оцінити необхідні інвестиції, керівництво компанії жахається від отриманих цифр. Це є наслідком саме надмірних вимог, начебто забезпечення доступності абсолютно всіх сервісів у режимі 24x7.

У таких випадках необхідно повернуться до стадії постановки завдання й виходити з реальних оцінок фінансових втрат бізнесу у випадку, якщо сервіс X буде недоступний у плині Y годин. Відштовхуючись від цієї інформації, потрібно розбити сервіси на кілька груп і для кожної визначити адекватні методи забезпечення відказостійкости й катастрофостійкості.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програмне забезпечення написане мовою Visual C#. Ця мова обрана виходячи з наступних міркувань. Visual C# – строго типізована об’єктно-орієнтована мова, призначена для розробки різноманітних безпечних і потужних додатків, виконуваних у середовищі .NET Framework. Мовою Visual C# можна розробляти звичайні клієнтські додатки Windows, веб-служби XML, розподілені компоненти, додатки типу “сервер-клієнт”, додатки баз даних і багато яких інших. В Visual C# 2012 є розширений редактор коду, конструктори зі зручним користувальницьким інтерфейсом, вбудований відладник і багато інших засобів,

покликані спростити розробку додатків мовою Visual C# версії 5.0 і .NET Framework версії 4.5.

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

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

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

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

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

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

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

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

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

Архітектура платформи .NET Framework

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

Вихідний код, написаний мовою Visual C#, компілюється в проміжну мову (IL) у відповідності зі специфікацією CLI. Код IL і ресурси, такі як растрові зображення й рядки, зберігаються на диску у файлі, що виконується, називаному складанням, з розширенням EXE або DLL у більшості випадків. Складання

містить маніфест із відомостями про типи складання, версії, мови й регіональні параметри та вимоги безпеки.

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

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

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

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи безперервної роботи додатків при серверній віртуалізації.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Представте, що ваш дата-центр (або сервер) сьогодні впав. Просто взяв і впав. Як показує практика, готові до цього далеко не всі:

– 93% компаній, які втрачали свій ЦОД на 10 і більше днів через катастрофу, стали банкрутами протягом року.

– Щотижня в США виходить із ладу 140 000 жорстких дисків.

– В 75% компаній немає рішень для аварійного відновлення.

– 34% компаній не тестують резервні копії.

– 77% тих, хто тестують, виявляли накопичувачі, які не читаються, у своїх бібліотеках.

Зараз поговоримо про технічні рішення, які в цьому допоможуть. Їхня вартість відрізняється від декількох тисяч до сотень тисяч доларів.

Висока доступність і аварійне відновлення

Дуже часто рішення для високої доступності (HA – High Availability) і аварійного відновлення (DR – Disaster Recovery) плутають. Насамперед, коли ми говоримо про безперервність бізнесу, ми маємо на увазі резервну площадку. Стосовно до IT – резервного ЦОД. Безперервність бізнесу – це не про резервне копіювання на бібліотеку в сусідній стійці (що теж дуже важливо). Це про те, що основний будинок компанії згорить, і ми через кілька годин або днів зможемо відновити роботу, розгорнувшись на новому місці:

Висока доступність:

– Рішення в межах одного ЦОД.

– Час відновлення < 30 хвилин.

– Нульова або близька до нуля втрата даних.

– Вимагає щоквартального тестування.

Аварійне відновлення:

– Містить у собі декілька віддалених ЦОД.
– Відновлення може займати години й навіть дні.
– Втрата даних може досягати багатьох годин.
– Вимагає щорічного тестування.

Виходить, потрібний резервний ЦОД. Які є варіанти? Звичайно виділяють три: гарячий, теплий і холодний резерви.

Холодний резерв

Холодний резерв має на увазі, що є якесь серверне приміщення, у яке можна завезти встаткування й розгорнути його там. При відновленні може плануватися закупівля «заліза», або його зберігання на складі. Потрібно враховувати, що більшість систем поставляється під замовлення, і швидко знайти десятки одиниць серверів, СЗД, комутаторів і інше буде нетривіальним завданням. Як альтернативу складуванню встаткування в себе, можна передбачити зберігання найбільш важливого або найбільш рідкого встаткування на складі ваших постачальників. При цьому, телекомунікаційні канали в приміщенні повинні бути присутнім, але висновок контракту із провайдером звичайно відбувається після ухвалення рішення про запуск «холодного» ЦОДа. Відновлення роботи в такому ЦОДі при катастрофічному збої основної площадки цілком може займати кілька тижнів. Переконаєтеся, що ваша компанія зможе проіснувати ці кілька тижнів без IT і не втратитися бізнесу (через відкликання ліцензії, або непоправного касового розриву, наприклад) – про це я писав раніше. Чесно говорячи, я б нікому не рекомендував цей варіант резервування. Можливо, я перебільшую роль IT у бізнесі деяких компаній.

Теплий резерв

Це значить, що в нас функціонує альтернативна площадка, у якій є активні інтернет і WAN-канали, базова телекомунікаційна й обчислювальна інфраструктура. Вона завжди «слабкіше» основний по обчислювальних потужностях, деяке встаткування там може бути відсутнім. Найважливіше – щоб

на площадці завжди була актуальна резервна копія даних. Діючи «по старинці» можна організувати регулярне переміщення туди резервних копій на стрічках. Сучасний метод – реплікація бекапів по мережі з основного ЦОДа. Використання бекапа з дедуплікацією дозволить оперативно передавати резервні копії навіть по «тонкому» каналі між ЦОДами.

Гарячий резерв

От він, вибір спеціалістів, що підтримують IT-системи, простій яких навіть на кілька годин приносить компанії величезні збитки. Тут є все необхідне встаткування для повноцінної роботи IT-систем. Звичайно фундаментом такої площадки служить система зберігання даних, на яку синхронно або асинхронно дзеркалюються дані з основного ЦОДа. Для того, щоб гарячий резерв у годину Ікс зміг відробити вкладені в нього гроші, повинні проводитися регулярні тестові переклади систем, налаштування й версія ОС серверів на основній і резервній площадці повинні постійно синхронізуватися – вручну або автоматично.

Мінус гарячого й теплого резерву – дороге встаткування простоює чекаючи катастрофи. Виходом із цієї ситуації є стратегія розподіленого ЦОДа. При такому варіанті дві (або більше) площадки рівноправні – більшість додатків можуть працювати як на одній, так і на іншій. Це дозволяє задіяти потужності всього встаткування й забезпечити балансування навантаження. З іншого боку, серйозно підвищуються вимоги до автоматизації перекладу IT-сервісів між ЦОДами. Якщо обоє ЦОДа «бойові», бізнес вправі очікувати, що при очікуваному піку навантаження на один з додатків, його можна швидко перевести в більше вільний ЦОД. Найчастіше, у подібних ЦОДах є присутнім синхронна реплікація між СЗД, але можлива й невелика асинхроність (у межах декількох хвилин).

Три чарівних слова

Перед тим, як перейти безпосередньо до технологій катастрофостійкості IT-сервісів, нагадаю три «чарівних» слова, які визначають вартість будь-якого DR-рішення: RTO, RPO, RCO:

– RTO (Recovery time objective) – час, за яке можливо відновити IT-систему

– RPO (Recovery point objective) – скільки даних буде загублено при аварійному відновленні

– RCO (Recovery capacity objective) – яку частину навантаження повинна забезпечувати резервна система. Цей показник може вимірятися у відсотках, транзакціях IT-систем і інших величин.

RPO

Перший розподіл, що ми можемо провести між всім різноманіттям IT-рішень для забезпечення катастрофостійкості – чи забезпечують вони нульове RPO чи ні. Відсутність втрати даних при збої забезпечується синхронною реплікацією. Найчастіше, це робиться на рівні СЗД, але можливо реалізувати й на рівні СУБД або сервера (за допомогою просунутого LVM). У першому випадку сервер не одержує від СЗД, з якої він працює, підтвердження про успішність запису, поки СЗД не передала цю транзакцію другій системі й не одержала від її підтвердження, що запис пройшов успішно.

Синхронну реплікацію вміють робити 100% СЗД, що відносяться до середнього цінового сегмента й деякі системи початкового рівня від відомих вендорів. Вартість ліцензій для синхронної реплікації на «простих» СЗД починається від декількох тисяч доларів. Приблизно стільки ж коштує софт для реплікації на рівні серверів на 2-3 сервери. Якщо у вас немає діючого резервного ЦОДа, не забудьте додати вартість закупівлі резервного встаткування.

RPO у кілька хвилин може забезпечити асинхронна реплікація на рівні СЗД, ПЗ керування томами сервера (LVM – Logical volume manager), або СУБД. Дотепер standby-копія бази даних залишається одним з найбільш популярних рішень для DR. Найчастіше функціонал “log shipping”, як це називається в адміністраторів СУБД, не ліцензується виробником окремо. Якщо у вас ліцензована БД – реплікуйте на здоров'я. Вартість асинхронної реплікації для серверів і СЗД не відрізняється від синхронної, див. попередній пункт.

Якщо ми говоримо про RPO у кілька годин, частіше це реплікація резервних копій з однієї площадки на іншу. Більшість дискових бібліотек уміють робити це, частина ПЗ для резервного копіювання – теж. Як я вже говорив, при такому варіанті здорово допоможе дедуплікація. Ви не тільки будете менше завантажувати канал передачею резервних копій, але й зробите це набагато швидше – кожний переданий бекап буде займати в десятки або сотні разів менше часу, чим у реальності. З іншого боку, треба пам'ятати, що перший бекап при дедуплікації однаково повинен передати в систему масу унікальних даних. «Справжню» дедуплікацію ви побачите після тижневого циклу резервного копіювання. При синхронізації дискових бібліотек – те ж саме. Якщо розрахунковий час передачі при вашій ширині каналу між ЦОД становить кілька днів і навіть тижнів (що може й коштувати чимало), їсти зміст спочатку поставити другу бібліотеку поруч, виконати синхронізацію й відвезти її в резервний ЦОД.

RTO

Коли стоїть завдання мінімізації часу відновлення (RTO), процес повинен бути максимально документований і автоматизований. Одне із кращих і найбільш універсальних рішень – HA-кластери з територіально рознесеними вузлами. Найчастіше, такі рішення будуються на базі реплікації СЗД, але можливі й інші варіанти. Лідируючі продукти в цій області, наприклад, Symantec Veritas Cluster, мають у своєму складі модулі по роботі зі СЗД, що перемикають напрямок реплікації, коли необхідно запустити знову сервіс на резервному вузлі. Для менш просунутих кластерів (наприклад Microsoft Cluster Services, убудований в Windows) основні виробники СЗД (IBM, EMC, HP) пропонують надбудови, роблячи зі звичайного HA-кластера катастрофостійкий.

Рідко хто замислюється про цікаву особливість переважною більшістю рішень по реплікації даних – їх «однозарядність». Ви можете одержати на резервній площадці тільки один стан даних. Якщо система із цими даними з якоїсь причини не стартувала – переходимо до плану «Б». Найчастіше це

відновлення з резервної копії з великою втратою даних. З перерахованих мною технологій виключення складе тільки реплікація тих же бекапів. Відповіддю тут є використання класу рішень Continuous Data Protection. Їхня суть у тім, що всі записи, що приходять від сервера, позначаються й зберігаються в певному журнальному томі на резервній площадці. При відновленні системи можна вибрати будь-яку точку із цього журналу й одержати стан не тільки на момент аварії, у якій дані були зіпсовані, але й за кілька секунд. Такі рішення захищають від внутрішньої погрози – видалення даних користувачами. У випадку реплікації СЗД – їй однаково, що передавати – порожній тім або вашу найбільш критичну БД. При використанні CDP можна вибрати момент прямо перед видаленням інформації й відновитися на нього. Вартість систем CDP звичайно – десятки тисяч доларів. Один з найбільш удалих прикладів, на мій погляд – EMC RecoverPoint.

Останнім часом набирають популярність системи віртуалізації СЗД. Крім своєї основної функції – об'єднання масивів різних вендорів у єдиний пул ресурсів – вони можуть сильно допомогти й в організації розподіленого ЦОДа. Суть віртуалізації СЗД у тім, що між серверами й системами зберігання з'являється проміжний шар контролерів, що пропускають крізь себе весь трафик. Тому зі СЗД презентуються не прямо серверам, а цим віртуалізаторам. Вони, у свою чергу, роздають їх хостам. У шарі віртуалізації можна робити реплікацію даних між різними СЗД, а найчастіше є й більше просунуті можливості – снепшоти, багаторівневе зберігання й т.д. При цьому сама базова функція віртуалізаторів є самою потрібною для цілей DR. Якщо в нас є дві СЗД у різних ЦОДах, з'єднаних оптичною магістраллю, ми беремо томи з кожної з них і збираємо «дзеркало» на рівні віртуалізатора. У підсумку ми одержуємо один віртуальний тім на два ЦОДа, що і бачать сервери. Якщо ці сервери віртуальні – починає працювати Live Migration віртуальних машин і можете «на ходу» переводити завдання між ЦОДами – користувачі нічого не помітять.

Повна втрата ЦОДа буде відпрацьована звичайним HA-кластером в автоматичному режимі за кілька хвилин. Мабуть, віртуалізація рознесених СЗД дозволяє забезпечити мінімальний час відновлення для більшості додатків. Для СУБД є неперевершений Oracle RAC і його аналоги, але вартість змушує задуматися. Віртуалізація SAN поки теж не дешева, для невеликих обсягів СЗД вартість рішення може бути менше $100К, але в більшості випадків ціна вище. На мій погляд, найбільш перевіреним рішенням є IBM SAN Volume Controller (SVC), найбільше технічно зробленим – EMC VPLEX.

До речі, якщо не всі ваші додатки ще живуть на віртуальному середовищі, варто спроектувати резервний ЦОД для них на віртуальних машинах. По-перше, вийде набагато дешевше, у других, зробивши це для резерву, недалеко й до міграції основних систем під керування якого-небудь гіпервізора.

Конкуренція на ринку аутсорсингу ЦОД робить більше вигідної оренду стійкомісць у ЦОДі провайдеру, у порівнянні з будівництвом і експлуатацією свого резервного центра. Якщо ви розміщаєте в нього віртуальну інфраструктуру, вийде серйозна економія на орендних платежах. Але й аутсорсингові ЦОДи вже не на вершині прогресу. Краще будувати резервну інфраструктуру відразу в «хмарі». Синхронізацію даних з основними системами при цьому можна забезпечити реплікацією на рівні сервера (є відмінне сімейство рішень DoubleTake від Vision Solutions).

Останній, але дуже важливий момент, про яке не можна забувати при проектуванні катастрофостійкої IT-інфраструктури – робочі місця користувачів. Те, що піднялася база даних, не означає поновлення бізнесу-процесу. Користувач повинен мати можливість виконувати свою роботу. Навіть повноцінний резервний офіс, у якому коштують виключені комп'ютери для ключових співробітників – не ідеальне рішення. У людини на втраченому робочому місці можуть бути довідкові матеріали, макроси, й інш., повноцінна робота без яких неможлива. Для найбільш важливих для компанії користувачів розумним виглядає перехід на віртуальні робочі місця (VDI). Тоді на робочому місці (будь

те звичайний ПК або модний «тонкий» клієнт) не зберігаються ніякі дані, він використовується тільки як термінал, щоб достукатися до Windows XP або Windows 7, що працює на віртуальній машині в ЦОДі. Доступ до такого робочого місця легко організувати з будинку або з будь-якого комп'ютера у філіальній мережі. Наприклад, якщо у вас кілька будинків і одне з них недоступно, ключові користувачі можуть приїхати в сусідній офіс і сісти на робочі місця «менш ключових». Потім вони спокійно логіняться в систему, попадають у свою віртуальну машину й фірма оживає!

У завершення, от основні питання, які варто задати при оцінці DR-рішення:

– Від яких збоїв захищає?
– Які RPO/RTO/RCO забезпечує?
– Скільки коштує?
– Наскільки складна експлуатація?

Катастрофостійких рішень незліченна безліч – як коробкових, так і тих, які можна зробити практично своїми руками. Будь ласка, поділитеся в коментарях що є у вас і історіями, як ці рішення вас виручали. Якщо у вас працює щось із описаних вище систем або їхніх аналогів – залишайте відкликання, наскільки спокійно ви спите, коли IT-системи під їхнім захистом.

3.2 Розробка структурної схеми

Для перезапуску додатків на сервері резервного ЦОДа потрібно об'єднати його із сервером основного ЦОДа, створивши географічно розподілений кластер (ще недавно ця функціональність підтримувалася тільки Unix/ RISC-серверами старшого класу й мейнфреймами). Крім того, для оперативного перемикання додатка з основного центра обробки даних на резервний в останньому повинні перебувати актуальні копії даних потрібних додатків, для чого необхідні спеціальні засоби віддаленої синхронної й асинхронної реплікації, які

використовуються в дискових масивах старшого класу (наприклад, EMC Symmetrix SRDF), а також орендовані або власні високошвидкісні канали зв'язку, що з'єднують обидві площадки. Зрозуміло, через більші витрати на створення й експлуатацію резервного центра обробки даних тільки самі великі організації можуть дозволити собі побудувати допоміжну площадку для захисту своїх бізнес-критичних додатків від катастроф.

У результаті застосування віртуалізації для консолідації серверних додатків і впровадження приватних хмар з'явилася потреба в новому підході до захисту від катастроф. При класичному способі захист забезпечувався тільки для декількох додатків, основні характеристики яких (вимоги до продуктивності сервера й системи зберігання) мало мінялися згодом. Очевидно, що старі методи Disaster Recovery, орієнтовані на фізичні сервери, не можна застосовувати до хмар, де можуть працювати десятки віртуалізированних додатків з різними вимогами до обчислювальної потужності і ємності. Крім того, для розгорнутих у приватній хмарі додатків потрібно забезпечити гнучке виділення ресурсів, а також можливість швидкого запуску нових додатків, у тому числі висококритичних і тому нужденних у захисті від катастроф. Нарешті, у більшості хмар як апаратна платформа використовуються сервери стандартної архітектури, які не підтримують класичні рішення географічно розподілених кластерів, розроблені для Unix-систем і мейнфреймів.

Призначене для віртуальних машин vSphere розроблене програмне забезпечення «Програмне забезпечення менеджера безперервної роботи додатків при серверній віртуалізації» на базі розробленої технології реплікації віртуальних машин «Програмне забезпечення реплікації баз даних» і розробленого пакета керування віртуальними машинами «Програмне забезпечення керування віртуальними машинами» реалізує автоматичне виконання планів післяаварійного відновлення з використанням заданих правил.

«Програмне забезпечення менеджера безперервної роботи додатків при серверній віртуалізації» координує роботу «Програмного забезпечення керування

віртуальними машинами», які обслуговують віртуалізированні сервери двох центрів обробки даних, тому, якщо з якоїсь причини в одному ЦОДі віртуальні машини зупиняються, відразу ж запускаються їхні копії в іншому. У плані післяаварійного відновлення визначається порядок вимикання й перезапуску віртуальних машин, вказуються виділювані їм ресурси в резервному центрі обробки даних і доступних мережних підключень. Крім того, плани післяаварійного відновлення можна створювати для окремих додатків і різних сценаріїв аварійних ситуацій.

«Програмне забезпечення реплікації баз даних» забезпечує незалежну від СЗД асинхронну реплікацію на рівні гіпервізора з можливістю налаштування цільових точок відновлення (RPO) і декількох точок відкату (point-in-time instance). Такий підхід дозволяє легко конфігурувати віртуальні машини vSphere без прив'язки до конкретної конфігурації фізичного сервера, набудовувати реплікацію на рівні дисків Virtual Machine Disk (VMDK) окремих віртуальних машин і проводити їхнє гранулярне відновлення.

Крім «рідної інтеграції» з «Програмним забезпеченням реплікації баз даних» (потрібно при використанні програмно обумовлених систем зберігання VSAN), «Програмне забезпечення менеджера безперервної роботи додатків при серверній віртуалізації» інтегрується з рішеннями по реплікації провідних виробників дискових масивів за допомогою спеціальних адаптерів. Його можна застосовувати й для тестування планів післяаварійного відновлення ключових додатків без переривання їхньої роботи. Тестування проводиться на тимчасових копіях реплікованих даних і в ізольованих мережах, тому функціонування робочих додатків обох центрів обробки даних не порушується. Крім того, це рішення дозволяє здійснювати безпечну міграцію додатків між ЦОДами за допомогою пакета «Програмне забезпечення безпечної міграції додатків між ЦОДами» і перерозподіляти навантаження між ними.

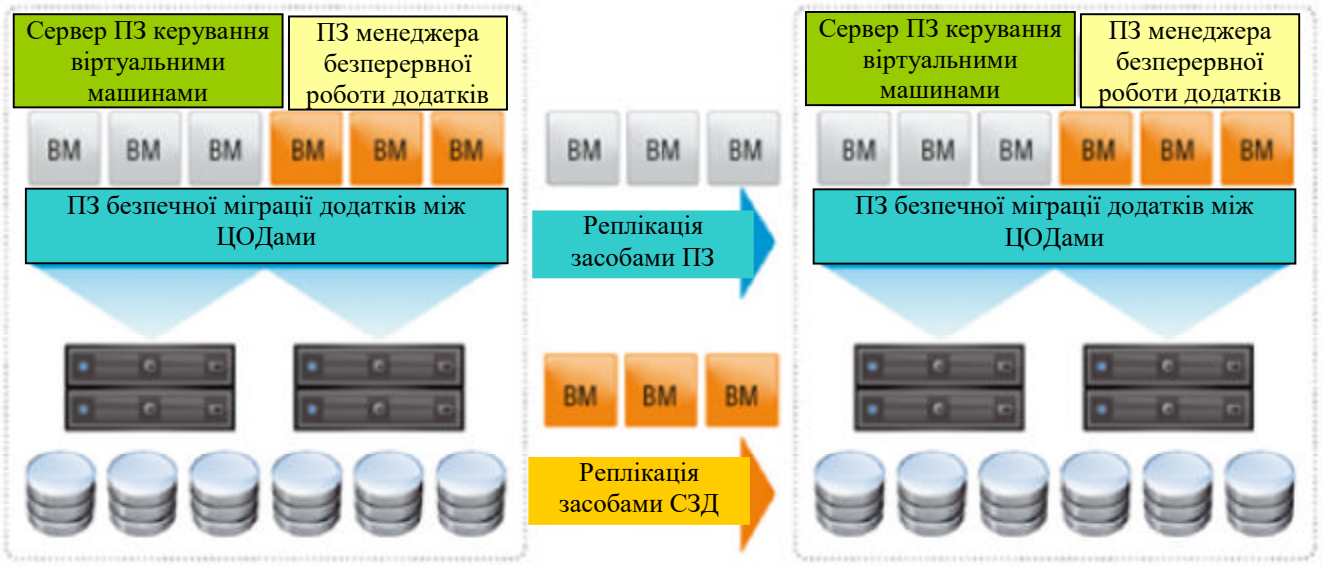

Рисунок 3.1 – Структурна схема системи

Післяаварійне відновлення як сервіс

При використанні розробленого програмного забезпечення «Програмне забезпечення менеджера безперервної роботи додатків при серверній віртуалізації», як і традиційних рішень захисту від катастроф, потрібний другий резервний центр обробки даних, територіально віддалений від основного. Більшість середніх по розмірі компаній, IT-бюджет яких невеликий, не мають фінансової можливості організувати другу площадку, але можуть впровадити катастрофостійке рішення за допомогою сервісів Disaster Recovery as Service (DRaaS), пропонованих провайдерами хмарних послуг. У моделі DRaaS роль резервного центра обробки даних виконує хмара провайдеру, і у випадку аварії на площадці замовника розташовані в ньому віртуальні машини і їхні додатки швидко перезапускаються.

Сервіс vCloud Air Disaster Recovery, якому можна розглядати як аналог «Програмне забезпечення менеджера безперервної роботи додатків при серверній віртуалізації» для DRaaS, призначений для аварійного відновлення віртуальних машин vSphere за допомогою гібридної хмари Air. Як і «Програмне забезпечення менеджера безперервної роботи додатків при серверній віртуалізації», він

припускає використання реплікації розробленого програмного забезпечення «Програмне забезпечення менеджера безперервної роботи додатків при серверній віртуалізації» і інструментів керування віртуальним середовищем «Програмне забезпечення керування віртуальними машинами» і дозволяє задавати цільову точку відновлення від 15 хв і до 24 год, причому можна створювати до 24 точок відкоту. У ньому передбачені можливості тестування плану післяаварійного відновлення для різних сценаріїв аварій. Передплата на vCloud Air Disaster Recovery забезпечує захист до 500 віртуальних машин.

У випадку Amazon Web Services (AWS) послуги DRaaS передбачають три варіанти рішення: «гаряче» (warm), коли цільовий час відновлення RTO не перевищує однієї години; «холодне» (cold) з RTO від одного робочого дня; Pilot-Light – проміжний варіант, при якому час відновлення становить не більше 4 ч. При холодному методі використовується традиційне резервне копіювання на стрічкові картриджі, що зберігаються в захищеному сховищі, при гарячому – резервується вся серверна інфраструктура замовника, а у варіанті Pilot-Light у хмарі AWS створюється пул ресурсів у мінімальній конфігурації, що при необхідності можна швидко розгорнути в повній конфігурації для обслуговування додатків замовника.

Microsoft пропонує у своїй хмарі Azure відновлення віртуальних машин vSphere і Hyper-V, а також додатків, що працюють на фізичних серверах. У сервісі Azure Site Recovery (ASR) використовуються функції реплікації SQL AlwaysON і Active Directory, забезпечується скорочення цільової точки відновлення до 30 сек. Клієнти ASR можуть виконувати перезапуск своїх віртуальних машин у хмарі Azure в автоматичному режимі й вручну по команді оператора. Після усунення аварії автоматичне поновлення роботи віртуальних машин у ЦОДі й зворотна реплікація виробляються автоматично.

Послуги захисту від катастроф за допомогою публічної хмари доступні й у декількох російських операторів ЦОДів. Улітку компанія DataLine розгорнула сервіс DRaaS на базі своєї хмари CloudLine. За допомогою рішення для

віддаленої реплікації Veeam програмне забезпечення, що розробляються, Replication користувачі цього сервісу можуть організувати реплікацію віртуальних машин у хмару CloudLine і настроїти перемикання на використання тиражованих копій при аварії у своєму центрі обробки даних. Поки реплікація в хмару реалізована для vSphere, але DataLine планує незабаром забезпечити захист і віртуальні машини Hyper-V.

У пропонованому системним інтегратором «Крок» сервісі DRaaS, створеному на базі побудованого в 2015 році хмари Cisco Powered, передбачені три варіанти забезпечення катастрофостійкості:

- перенос віртуальних машин у хмару;
- розгортання в хмарі нових серверів;
- використання хмари як резервного ЦОДа.

Компанія IT-Grad у своїй публічній хмарі реалізувала сервіс «Резервна площадка» для тих замовників, які або вже розгорнули власну віртуальну інфраструктуру, або планують використовувати технології віртуалізації на базі vSphere. Відказостійкість у цьому рішенні досягається за допомогою засобів vShpere і vCenter «Програмне забезпечення менеджера безперервної роботи додатків при серверній віртуалізації», а також технологій дзеркалювання SnapMirror і систем зберігання NetApp FAS. Для «Програмне забезпечення реплікації баз даних» цільова точка відновлення RPO становить від 15 хв до 24 год, а у випадку застосування NetApp SnapMirror цей показник RPO зменшується майже до нуля.

Звертання до сервісів DRaaS дозволяє малому й середньому бізнесу реалізувати за допомогою хмарних технологій захист від катастроф для своїх основних додатків без організації резервного центра обробки даних і застосування систем зберігання корпоративного класу.

3.3 Розробка функціональної схеми

Для реалізації системи безперервної роботи додатків при серверній віртуалізації використовується технологія RAID. Принцип функціонування RAID-системи полягає в наступному: з набору дискових накопичувачів СХД створюється масив, що управляється спеціальним контролером і визначається комп'ютером як єдиний логічний диск великої ємності. За рахунок паралельного виконання операцій вводу-виводу забезпечується висока швидкодія системи, а підвищена надійність зберігання інформації досягається дублюванням даних або обчисленням контрольних сум. Слід зазначити, що застосування RAID-масивів захищає від втрат даних тільки у випадку фізичної відмови жорстких дисків.

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

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

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

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

Переваги:

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

– простота реалізації;

– низька вартість;

– максимальна ефективність використання дискового простору – 100%.

Недоліки:

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

– відмова одного диска спричиняє втрату всіх даних масиву.

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

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

RAID 1 – найпростіший відказостійкий масив.

Переваги:

– простота реалізації;

– простота відновлення масиву у випадку відмови (копіювання).

Недоліки:

– висока вартість – 100-процентна надмірність;

– невисока швидкість передачі даних.

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

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

зберігання контрольної інформації. Їхнє число дорівнює кількості біт у слові контрольної суми.

RAID 2 не одержав комерційного застосування.

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

Переваги:

– досить проста реалізація;

– корекція помилок "на льоту";

– дуже висока швидкість передачі даних;

– при збільшенні кількості дисків накладні витрати зменшуються.

Недоліки:

– низька швидкість обробки запитів;

– висока вартість;

– більша надмірність.

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

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

В RAID 3 інформація розбивається на порції однакового розміру.

Цей рівень має набагато меншу надмірність, чим RAID 2. У RAID 2 більшість дисків, що зберігають контрольну інформацію, потрібні для визначення несправного розряду. Як правило, RAID-контролери можуть одержати дані про

помилку за допомогою механізмів відстеження випадкових збоїв. За рахунок розбивки даних на порції RAID 3 має високу продуктивність. Оскільки при кожній операції вводу/виводу виробляється обіг практично до всіх дисків масиву, те одночасна обробка декількох запитів неможлива.

Цей рівень підходить для додатків з файлами великого обсягу й малою частотою обігів (в основному це сфера мультимедіа). Використання тільки одного диска для зберігання контрольної інформації пояснює той факт, що коефіцієнт використання дискового простору досить високий (як наслідок цього – відносно низька вартість). Для реалізації масиву потрібно не менше трьох вінчестерів.

Переваги:

– відмова диска мало впливає на швидкість роботи масиву;

– висока швидкість передачі даних;

– високий коефіцієнт використання дискового простору.

Недоліки:

– складність реалізації;

– низька продуктивність при великій інтенсивності запитів даних невеликого обсягу.

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

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

В RAID 4 потік даних розділяється на блоки.

В RAID 4 можливо одночасне виконання декількох операцій читання. Цей масив підвищує продуктивність передачі файлів малого обсягу (за рахунок розпаралелювання операції зчитування). Але оскільки при записі повинна змінюватися контрольна сума на виділеному диску, одночасне виконання

операцій неможливо (у наявності асимметричність операцій вводу й виводу). Цей рівень має майже всі недоліки RAID 3 і не забезпечує переваги у швидкості при передачі даних великого обсягу. Схема зберігання розроблялася для додатків, у яких дані споконвічно розбиті на невеликі блоки, тому немає необхідності розбивати їх додатково. Ця схема зберігання даних має невисоку вартість, але її реалізація досить складна, як і відновлення даних при збої.

Переваги:

– висока швидкість передачі даних;

– відмова диска мало впливає на швидкість роботи масиву;

– високий коефіцієнт використання дискового простору.

Недоліки:

– досить складна реалізація;

– дуже низька продуктивність при записі даних;

– складне відновлення даних.

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

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

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

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

Найбільший недолік рівнів RAID від 2-го до 4-го – це наявність окремого диска (або дисків), що зберігає інформацію про парність. Швидкість виконання операцій зчитування досить висока, тому що не вимагає звертання до цього

диска. Але при кожній операції запису на ньому змінюється інформація, тому схеми RAID 2-4 не дозволяють проводити паралельні операції запису. RAID 5 не має цього недоліку, тому що контрольні суми записуються на всі диски масиву, що уможливлює виконання декількох операцій читання або записи одночасно. RAID 5 має досить високу швидкість запису/читання й малу надмірність.

Переваги:

– висока швидкість запису даних;

– досить висока швидкість читання даних;

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

– високий коефіцієнт використання дискового простору.

Недоліки:

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

– досить складна реалізація;

– складне відновлення даних.

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

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

RAID 6 використовує дві незалежні схеми контролю парності.

Цей рівень має дуже високу відказостійкість, більшу швидкість зчитування (дані зберігаються блоками, немає виділених дисків для зберігання

контрольних сум). У той же час через великий обсяг контрольної інформації RAID 6 має низьку швидкість запису. Він дуже складний у реалізації, характеризується низьким коефіцієнтом використання дискового простору: для масиву з п'яти дисків він становить усього 60%, але з ростом числа дисків ситуація виправляється.

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

Переваги:

– висока відказостійкість;

– досить висока швидкість обробки запитів;

Недоліки:

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

– дуже складна реалізація;

– складне відновлення даних;

– низька швидкість запису даних.

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

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

RAID 7 – зареєстрована торговельна марка компанії Storage Computer Corporation.

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

Переваги:

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

– гарна масштабованість;

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

– відсутність необхідності в додатковій передачі даних для обчислення парності.

Недоліки:

– власність однієї компанії;

– складність реалізації;

– дуже висока вартість на одиниці об'єму;

– не може обслуговуватися користувачем;

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

– короткий гарантійний строк.

RAID 10

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

Комбіновані рівні

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

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

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

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

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

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

Matrix RAID

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

Програмний RAID

Для реалізації RAID можна застосовувати не тільки апаратні засоби, але й повністю програмні компоненти (драйвери). Наприклад, у системах на ядрі Linux існують спеціальні модулі ядра, а управляти RAID-пристроями в GNU/Linux

можна за допомогою утиліти mdadm. Програмний RAID має свої достоїнства й недоліки. З одного боку, він нічого не коштує (на відміну від апаратних RAID-контролерів, ціна яких від $250). З іншого боку, програмний RAID використовує ресурси центрального процесора, і в моменти пікового навантаження на дискову систему процесор може значну частину потужності витрачати на обслуговування RAID-пристроїв.

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

Система складається з наступних блоків.

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

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

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

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

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

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

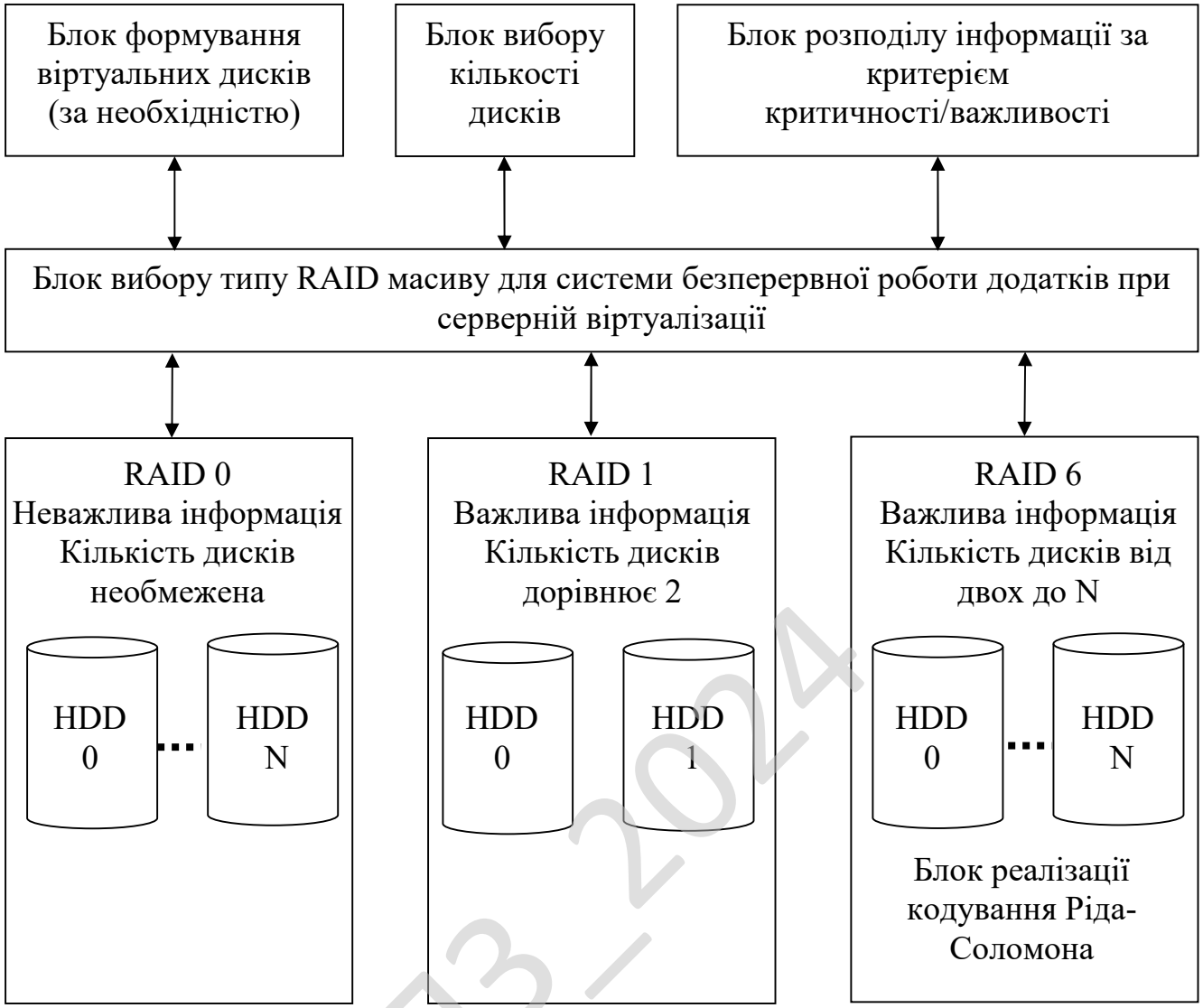

Рисунок 3.2 – Функціональна схема системи

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

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

Коди Ріда-Соломона є важливим частковим випадком БЧХ-коду, корінь полінома, що породжує, якого лежать у тім же полі, над яким і будується код

(m = 1).

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

Поліном, що породжує

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

Теорема 3.1

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

Теорема 3.2

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

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

Матриця, що породжує

Поліноми $g(x), xg(x), x^2 g(x), \ldots, x^{k-1} g(x)$ 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, т.е. $\alpha^{q-1}=1$, $\alpha^i \neq 1$, $0 < i < q-1$.

Тоді нормований поліном g(x) мінімального ступеня над полем GF(q), коріннями якого є d – 1 підряд, що йдуть ступенів, $\alpha^{l_0}, \alpha^{l_0+1}, \dots, \alpha^{l_0+d-1}$ 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) ≤ 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 $\ln n^2$.

Кодування

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

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

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

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

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

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

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

Результатом операції є результат додавання, перетворена обернено в

поліноміальну форму.

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

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

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

На структурній схемі кодера (рисунок 3.2) символу «С» відповідають дві константи GF(n).

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

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

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

Декодування

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

– Обчислює синдром помилки.

– Будує поліном помилки.

– Знаходить корінь даного полінома.

– Визначає характер помилки.

– Виправляє помилки.

Обчислення синдрому помилки

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

Побудова полінома помилки

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

Поліном помилок реалізується за допомогою алгоритму Берлекемпа-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 нацело, тобто

$$a = bq_0 + r_1$$

$$b = r_1 q_1 + r_2$$
$$r_1 = r_2 q_2 + r_3 \quad (3.3)$$
$$\dots$$
$$r_{n-1} = r_n q_n$$

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

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

Коректність цього алгоритму випливає з наступних двох тверджень:

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

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

Знаходження корня

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

Визначення характеру помилки і її виправлення

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

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

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється. Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

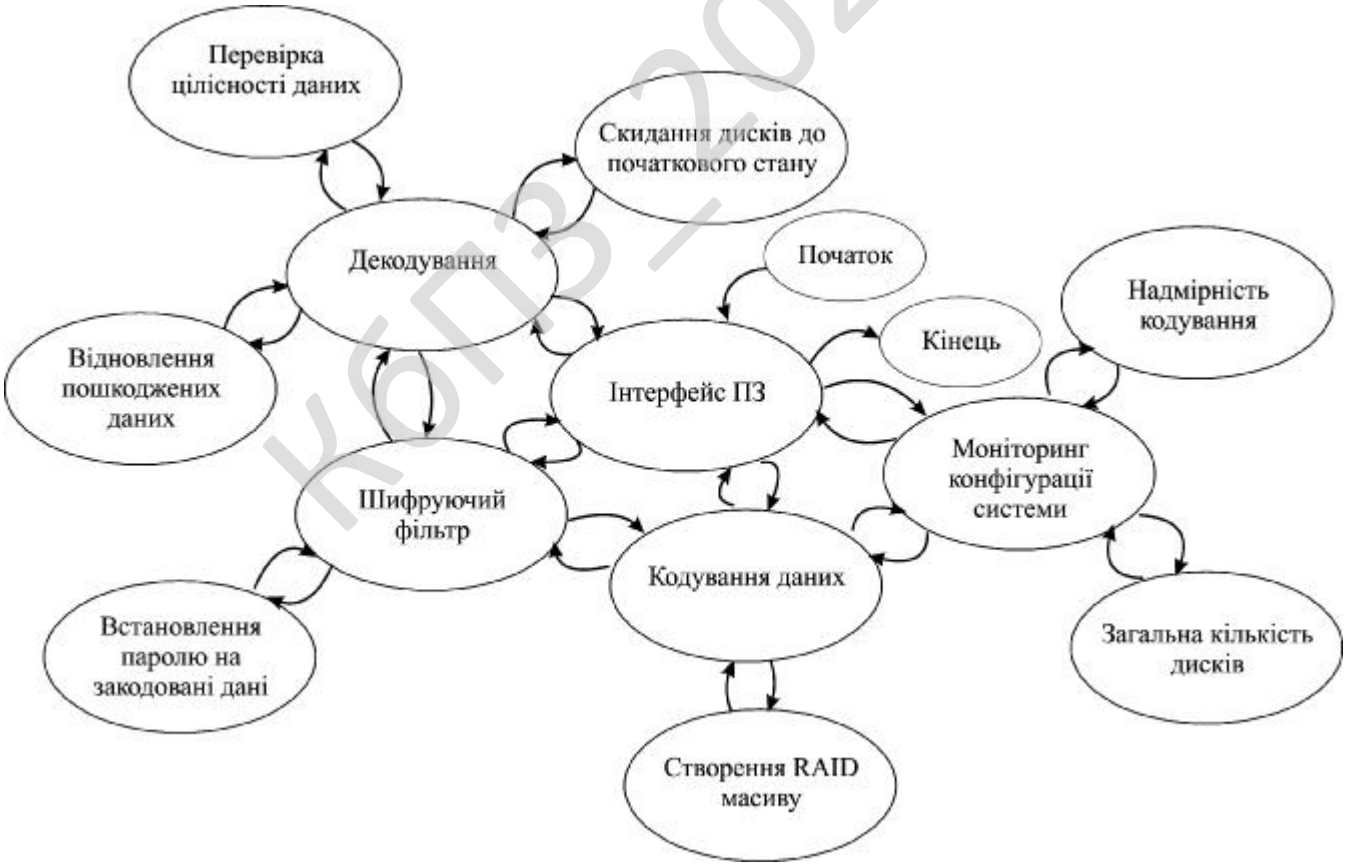

Рисунок 3.3 – Діаграма взаємодії процесів

Діаграми потоків даних містять чотири типи елементів:

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

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

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

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

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

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

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

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірки поточного стану та поверненням на початок схеми чи з завершенням роботи розробленого ПЗ.

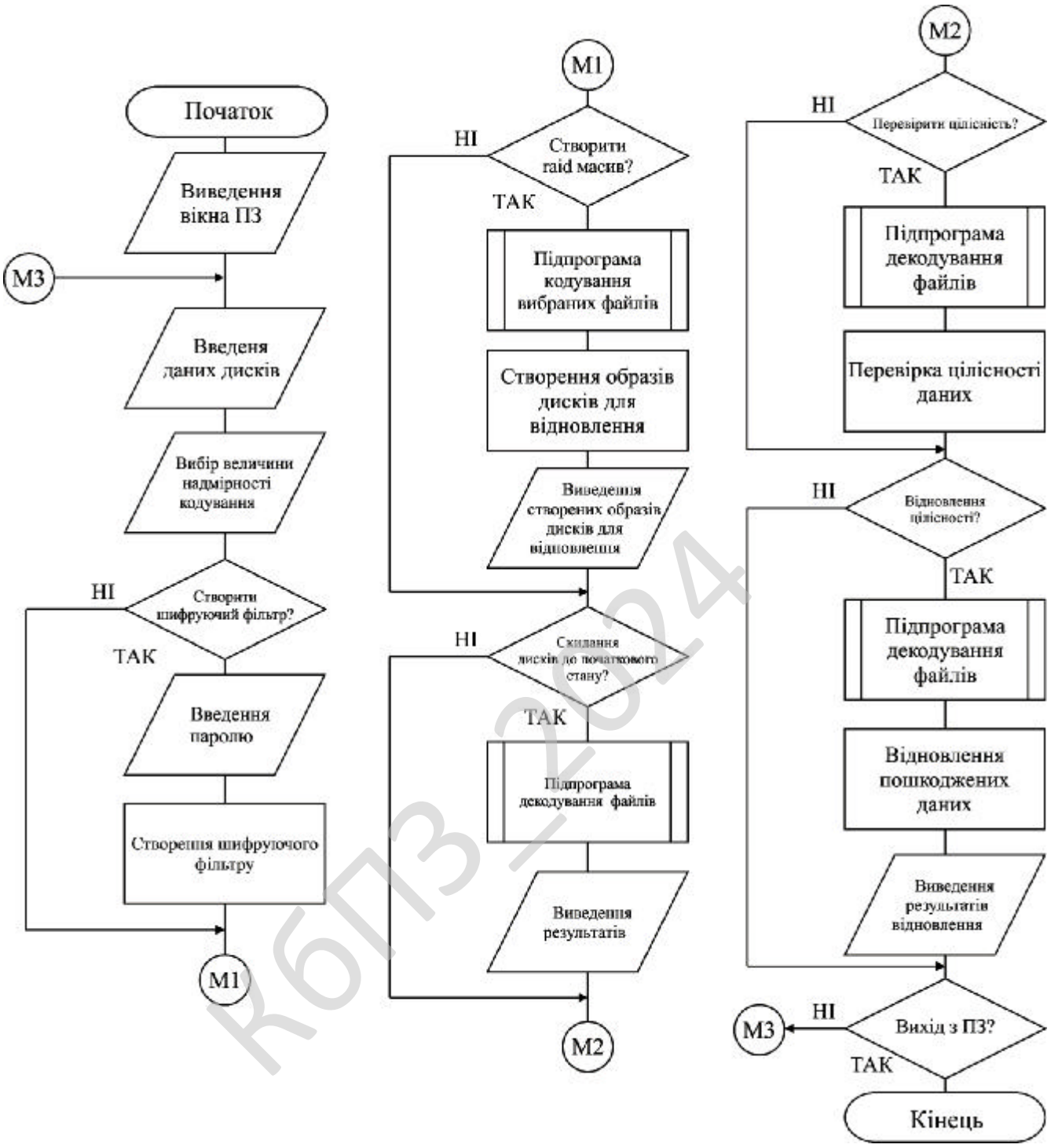

Рисунок 4.1 – Блок-схема основної програми

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

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю.

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

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

Управління вимогами це процес запису, аналізу, трасування, пріоритезації і узгодження вимог та контролю змін і доведення до їх зацікавлених сторін. Це безперервний процес протягом всього життя проекту. Вимога – якість, якій мають відповідати результати проекту (продукту або послуги).

Мета управління вимогами полягає в тому, щоб переконатися, що організація відповідає потребам і очікуванням своїх клієнтів, внутрішніх або зовнішніх зацікавлених сторін. Управління вимогами починається з аналізу і виявлення цілей і обмежень організації. Управління вимогами додатково включає в себе підтримку планування вимог, інтеграції вимог і організації роботи з ними (атрибути для вимог). Управління вимогами передбачає спілкування між членами проектної групи і зацікавленими сторонами, і адаптацію до змін у вимогах протягом всього проекту. Щоб запобігти перетину поля одного класу вимог з іншим, постійні зв'язки між членами команди розробників є критичними. Наприклад, при розробці програмного забезпечення для внутрішнього використання у бізнесу можуть бути настільки сильні потреби, що він може проігнорувати вимоги користувачів, або вважати, що створені сценарії використання покриють також і користувальницькі вимоги.

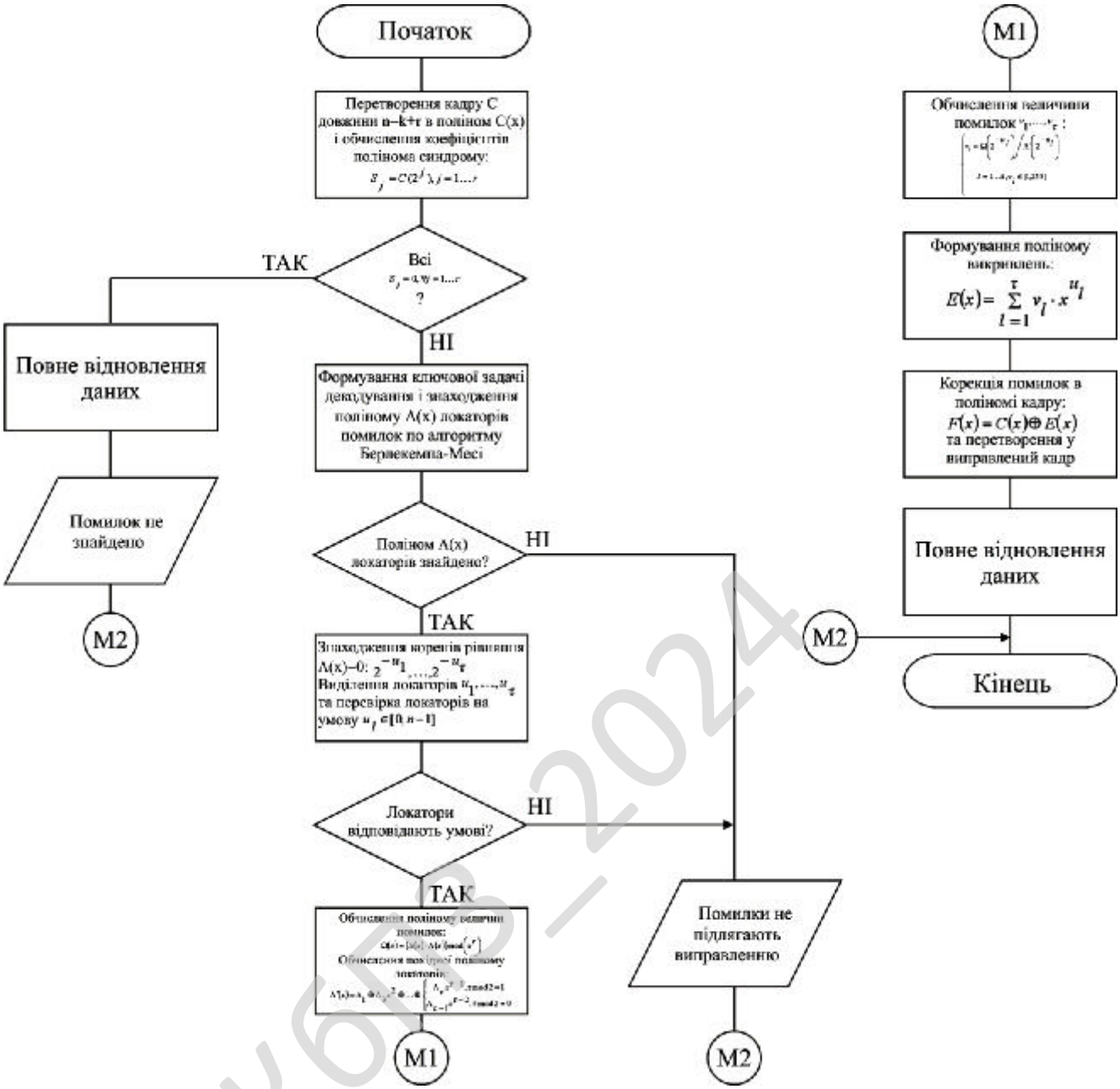

Рисунок 4.2 – Блок-схема роботи підпрограми

Відслідковування вимоги фактично означає документування всього життєвого циклу вимоги. Часто необхідно дізнатися першоджерело кожної вимоги. Для цього всі зміни вимог повинні бути задокументовані, щоб досягти стану повного відстеження. Відстежувати треба бути навіть використання реалізованих вимог.

Вимоги мають різні джерела, такі як ділова людина, що замовляє продукт, менеджер зі збуту і фактичний користувач. У всіх цих людей є різні вимоги до продукту. Використовуючи відслідковування вимог, реалізована в системі функція може бути простежена назад до людини або групі, яка замовляла її під час збору вимог. Ця особливість може, наприклад, використовуватися в процесі розробки для пріоритезації вимог, визначаючи, наскільки цінною є дана вимога для певного користувача.

Відслідковування може також використовуватися після розгортання продукту. Наприклад, коли вивчення використання системи показує, що якась функція не використовується, можна визначити навіщо вона була потрібна спочатку.

Завдання управління вимогами.

На кожному етапі процесу розробки існують ключові методи і задачі пов'язані з управлінням вимогами. Для ілюстрації, розглянемо наприклад стандартний процес розробки з п'ятьма фазами: дослідженням, аналізом здійсненності, дизайном, розробкою та тестуванням і випуском.

Дослідження. Під час фази дослідження збираються перші три класи вимог від користувачів, бізнесу і команди розробників. У кожній області задають однакові питання: які цілі, які обмеження, які використовуються процеси та інструменти і так далі. Тільки коли ці вимоги добре зрозумілі, можна приступати до розробки функціональних вимог.

Тут необхідне застереження: незалежно від того, як сильно група намагається це зробити, вимоги не можуть бути повністю визначені на початку проекту. Деякі вимоги змінюються, або тому що вони просто не були знайдені спочатку, або тому що внутрішні чи зовнішні сили торкаються проекту в середині циклу. Таким чином, учасники групи повинні спочатку погодитися, що головна умова успіху – гнучкість у мисленні та діях.

Результатом стадії дослідження є документ – специфікація вимог, схвалений усіма членами проекту. Пізніше, в процесі розробки, цей документ

буде важливий для запобігання розповзання меж проекту або непотрібних змін. Оскільки система розвивається, кожна нова функція відкриває світ нових можливостей, таким чином специфікація вимог прив'язує команду до оригінального бачення системи і дозволяє контрольоване обговорення змін.

У той час як багато організацій все ще використовують звичайні документи для керування вимогами, інші управляють своїми базовими вимогами, використовуючи програмні інструменти.

Ці інструменти керують вимогами використовуючи базу даних, і зазвичай мають функції автоматизації відстеження (наприклад, дозволяючи створювати зв'язки між батьківськими і дочірніми вимогами, або між тестами і вимогами), управління версіями, і управління змінами. Зазвичай такі інструментальні засоби містять функцію експорту, яка дозволяє створювати звичайний документ, експортуючи дані вимог.

Аналіз здійсненності.

На стадії аналізу здійсненності визначається вартість вимог. Для користувальницьких вимог поточна вартість роботи порівнюється з майбутньою вартістю встановленої системи. Задаються питання такі як: «Скільки нам зараз варті помилки введення даних?» Або, «Яка вартість втрати даних через помилки оператора пов'язаної з використовуваним інтерфейсом?». Фактично, потреба в новому інструменті часто розпізнається, коли подібні питання потрапляють до уваги людей, що займаються в організації фінансами.

Ділова вартість включає відповіді на такі питання як: «У якого відділу є бюджет на це?» «Який рівень повернення коштів від нового продукту на ринку?» «Який рівень скорочення внутрішніх витрат на навчання і підтримку, якщо ми зробимо нову, більш просту в використанні систему?»

Технічна вартість пов'язана з вартістю розробки програмного забезпечення та апаратною вартістю. «Чи є у нас потрібні люди, щоб створити інструмент?» «Чи потребуємо ми нове устаткування для підтримки нової системи?»

Подібні питання дуже важливі. Група повинна з'ясувати, чи буде новий автоматизований інструмент мати достатню ефективність аби перенести частину тягаря користувачів на систему і зекономити час людей.

Ці питання також вказують на основну суть управління вимогами. Людина і інструмент формують систему, і це розуміння особливо важливе, якщо інструмент – комп'ютер або новий додаток на комп'ютері. Людський розум вкрай ефективний у паралельній обробці та інтерпретації тенденцій з недостатніми даними. Комп'ютерний процесор ефективний у послідовній обробці і точному математичному обчисленні. Основна мета управління вимогами для програмного проекту полягала б у тому, щоб гарантувати, що автоматизована робота призначена «правильному» процесору.

Наприклад, «не змушуйте людину пам'ятати, де вона знаходиться в системі. Примусьте інтерфейс завжди повідомляти про місцезнаходження людини в системі ». Або «не змушуйте людини вводити ті ж самі дані в два екрани. Примусьте систему зберігати дані і заповнювати їх де необхідно автоматично». Результатом стадії аналізу здійсненності є бюджет і графік проекту.

Дизайн. Припускаючи, що вартість точно визначена і переваги, які будуть отримані, є досить великими, проект може перейти до стадії проектування.

На стадії дизайну основна діяльність управління вимогами полягає в тому, щоб перевіряти чи відповідають результати дизайну документу вимог, щоб упевнитися, що робота залишається в межах проекту. І знову, гнучкість є ключем до успіху. Ось класичний приклад змін проекту, які відмінно працювали. Проектувальники Форда на початку 1980-х очікували, що ціни на бензин піднімуться до 3,18 дол за галон до кінця десятиліття. На середині процесу дизайну автомобіля Ford Taurus, ціни встановилися приблизно на рівні 1,50 дол за галон. Колектив дизайнерів вирішив, що вони могли б створити більший, більш зручний, і більш потужний автомобіль, якщо б ціни на бензин залишилися низькими. Таким чином, вони перепроектувати автомобіль. Коли новий автомобіль вийшов, він встановив загальнонаціональні рекорди продажів.

У більшості випадків, однак, відступ від оригінальних вимог до такої міри не працює. Таким чином документ вимог стає ключовим інструментом, який допомагає команді приймати рішення про зміни дизайну.

Розробка та тестування. На стадії розробки і тестування, основна діяльність управління вимогами – це гарантувати, що робота і ціна залишаються в межах графіка і бюджету, і що створюваний інструмент дійсно відповідає вимогам. Основним інструментом, використовуваним на цій стадії, є створення прототипу і ітераційне тестування. Для програмного додатка користувацький інтерфейс може бути створений на папері і перевірений з потенційними користувачами, в той час як створюється основа програми. Результати цих тестів записуються в керівництві по дизайну користувацького інтерфейсу і передаються колективу дизайнерів. Це економить їх час і робить їх завдання набагато простіше.

```
/*---------------------------------
* кодер Ріда-Соломона
-----------------------------------*/
encode_rs()
{
        int i, j;
        int feedback;
        // ініціалізація поля біт парності нулями
        for (i = 0; i < n - k; i++) b[i] = 0;
        // обробляємо всі символи
        // вихідних даних праворуч ліворуч
        for (i = k - 1; i >= 0; i-i--)
        {
// готовимо (data[i] + b[n - k - 1]) до множення на g[i]
// тобто складаємо черговий "захоплений" символ вихідних
// даних з молодшим символом біт парності (відповідного
// "регістру" b2 t-1, (рисунок 4.2) і переводимо його в
// індексну форму, зберігаючи результат у регістрі feedback;
// як ми вже говорили, сума двох індексів є добуток  поліномів
                feedback = index_of[data[i] ^ b[n - k - 1]];
 // є ще символи для обробки?
                if (feedback != -1)
                {
```

```
 // здійснюємо зрушення ланцюга bx-регістрів
                    for (j = n - k - 1; j > 0; j-j--)
  // якщо поточний коефіцієнт g - це дійсний
 // (тобто ненульовий коефіцієнт, те
  // множимо feedback на відповідний g-коефіцієнт
  // і складаємо його з наступних елементів ланцюжка
   if (g[j] != -1) b[j] = b[j - 1] ^ alpha_to[(g[j] + feedback) % n];
                    else
   // якщо поточний коефіцієнт g - це нульовий коефіцієнт,
   // виконуємо одне лише зрушення без множення, переміщаючи
   // символ з одного m-регістра в інший
                          b[j] = b[j - 1];
   // закільцьовуємо вихідний символ у крайній лівий b 0-регістр
                    b[0] = alpha_to[(g[0] + feedback) % n];
               }
               else
               {
  // розподіл завершений,
  // здійснюємо останнє зрушення регістра,
  // на виході регістра буде частка, що губиться,
  // а в самому регістрі - шуканий залишок
for (j = n - k - 1; j > 0; j-j--) b[j] = b[j - 1]; b[0] = 0;
               }
       }
}
```

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.

При розробці ПЗ було використано підходи ри́зик-ме́неджменту – це система управління ризиками, яка включає в себе стратегію та тактику управління, направлені на досягнення основних цілей. Ефективний ризик-менеджмент включає:

– систему управління;

– систему ідентифікації і вимірювання;

– систему супроводження (моніторингу та контролю).

Сучасна наука представляє ризик як вірогідну подію, в результаті настання якої можуть відбутися позитивні, нейтральні або негативні наслідки. Якщо ризик припускає наявність як позитивних, так і негативних результатів, він відноситься

до спекулятивних ризиків. Якщо ж наслідки негативні, або відсутні взагалі, такий ризик іменується чистим.

Мета ризик-менеджменту – підвищення конкурентоспроможності господарюючих суб'єктів за допомогою захисту від реалізації чистих ризиків.

Теорія ризик-менеджменту ґрунтується на трьох базових поняттях: корисності, регресії і диверсифікації. У 1738 швейцарський математик Даніель Бернуллі доповнив теорію вірогідності методом корисності або привабливості того або іншого результату подій. Ідея Бернуллі полягала в тому, що в процесі ухвалення рішення люди приділяють більше уваги розміру наслідків різних результатів, ніж їх вірогідність. В кінці XIX століття англійський дослідник Ф. Гальтон запропонував вважати регресію або повернення до середнього значення універсальною статистичною закономірністю. Суть регресії трактувалася ним як повернення явищ до норми з часом. Згодом було доведено, що правило регресії діє в найрізноманітніших ситуаціях, починаючи з азартних ігор та розрахунку вірогідності виникнення нещасних випадків, і закінчуючи прогнозуванням коливань економічних циклів. У 1952 аспірант Університету Чикаго Гарі Марковіц в статті «Диверсифікація вкладень» («Portfolio Selection») математично обґрунтував стратегію диверсифікації інвестиційного портфеля, зокрема, він показав, як шляхом продуманого розподілу вкладень мінімізувати відхилення прибутковості від очікуваного показника. У 1990 Г. Марковіцу присуджена Нобелівська премія за розробку теорії і практики оптимізації портфеля фондових активів.

Етапи ризик-менеджменту.

У ризик-менеджменті прийнято виділяти декілька ключових етапів:

– на першому етапі відбувається виявлення ризику з супутньою оцінкою вірогідності його реалізації і масштабу наслідків;

– на другому етапі здійснюється розробка ризик-стратегії з метою зниження вірогідності реалізації ризику і мінімізації можливих негативних наслідків;

– на третьому етапі вибираються методи і інструменти управління виявленим ризиком;

– на четвертому етапі проводиться безпосереднє управління ризиком;

– на завершальному етапі оцінюються досягнуті результати і коректується ризик-стратегія.

За ключовий етап ризик-менеджменту вважається етап вибору методів і інструментів управління ризиком.

Методи і інструментарій ризик-менеджменту.

Базовими методами ризик-менеджменту є відмова від ризиків, зниження, передача і ухвалення.

Ризик-інструментарій значно ширший. Він включає політичні, організаційні, правові, економічні, соціальні інструменти, причому ризик-менеджмент як система допускає можливість одночасного застосування декількох методів і інструментів ризик-управління.

Найбільш часто вживаним інструментом ризик-менеджменту є страхування. Страхування припускає передачу відповідальності за відшкодування передбачуваного збитку сторонній організації (страхової компанії).

Прикладами інших інструментів можуть бути відмова від надмірно ризикової діяльності (метод відмови), профілактика або диверсифікація (метод зниження), аутсорсинг витратних ризикових функцій (метод передачі), формування резервів або запасів (метод ухвалення).

Хоча я реалізовував програму сам, було використано підходи Scrum для саморозвитку та пришвидшенню розробки, розглянемо цей метод. Scrum – підхід управління проектами для гнучкої розробки програмного забезпечення. Скрам чітко робить акцент на якісному контролі процесу розробки.

Підхід вперше описали Гіротака Такеучі та Ікуджіро Нонака в статті The New New Product Development Game (Гарвардський Діловий Огляд, січ–лют 1986). Вони відзначили, що проекти, над якими працюють невеликі, крос–функціональні команди, зазвичай систематично продукують кращі результати, і

пояснили це, як «підхід регбі». У 1991 році ДеҐрейс та Шталь у книжці Злі проблеми, справедливі рішення послалися на цей підхід, як на Scrum (штовханина; сутичка навколо м'яча (у регбі)), спортивний термін, згаданий в статті Такеучі і Нонака. Кен Швабер на початку 1990–х використовував підхід який привів Scrum в його компанію.

Вперше метод Scrum було представлено на загальний огляд задокументованим, чітко сформульованим та описаним спільно Сазерлендом та Швабером на OOPSLA'96 в Остіні. Швабер та Сазерленд протягом наступних років працювали разом щоб обробити та описати весь їхній досвід та найкращі практичні зразки для індустрії в одне ціле, в ту методологію, що відома сьогодні як Scrum. Швабер об'єднав зусилля з Майком Бідлом в 2001, щоб детально описати метод в книжці Agile Software Development with SCRUM. Не зважаючи на те, що для Scrum нарікли долю управління проектами з розробки ПЗ, він може також використовуватися в роботі команд обслуговувань програмного забезпечення (software maintenance teams), або як підхід управління розробкою і супроводом програм: Scrum of Scrums.

Scrum – це кістяк процесу, який включає набір методів і попередньо визначених ролей. Головні дійові особи – ScrumMaster, той хто опікується процесами, веде їх і працює як керівник проекту, Власник Продукту, людина, що представляє інтереси кінцевих користувачів та інших зацікавлених в продукті сторін, та Команду, яка включає розробників.

Протягом кожного спринту, 15–30 денного періоду (тривалість визначається командою), працівники створюють функціональний ріст програмного забезпечення. Набір можливостей, які імплементуються кожного спринту, приходять з етапу, що має назву product backlog (документація запитів на виконання робіт), який має найвищу пріоритетність за рівнем вимог до роботи, що повинна бути виконана.

Запити на виконання робіт (backlog items), що визначені протягом наради з планування спринту (sprint planning meeting), переміщуються в етап спринту.

Протягом цієї наради Власник Продукту інформує про завдання, які він хоче, аби були виконані. Тоді Команда визначає, скільки з бажаного вони можуть зробити, щоб завершити необхідні частини протягом наступного спринту. Протягом спринту команда виконує визначений фіксований список завдань (т.з. backlog items). Впродовж цього періоду ніхто не має права змінювати перелік запитів на виконання робіт, що слід розуміти, як заморожування вимог (requirements) протягом спринту. Product backlog – це документ, який має список вимог до функціональності, які упорядковані згідно зі ступенем важливості. Product backlog представляє список того, що повинно бути реалізовано. Елементи цього списку називається «історіями» (user story) або елементами backlog–у (backlog items). Product backlog відкритий для редагування усім учасникам Scrum–процесу.

Обов'язкові поля:

1. ID – унікальний ідентифікатор, порядковий номер, який використовується для ідентифікації історій у разі їх перейменування.

2. Назва (Name) – стислий опис історії. Він повинен бути однозначним, щоб і розробники і product owner могли зрозуміти, про що йдеться і відрізнити одну історію від іншої.

3. Важливість (Importance) – ступінь важливості даної історії на погляд product owner ’а. Зазвичай являє собою натуральне число, іноді для цієї цілі використовуються числа Фібоначчі. Чим більше значення, тим більше пріоритет.

4. Попередня оцінка (initial estimate) – початкова оцінка об'єму робіт, необхідного для реалізації історії порівняно з іншими історіями. Вимірюється у story point'ах. Приблизно відповідає числу «ідеальних людино–днів».

5. Як продемонструвати (how to demo) – стисле пояснення того, як завершена задача буде продемонстрована у кінці спринта. Дане поле може являти собою код автоматизованого приймального тесту.

Додаткові поля. Іноді, також, використовуються додаткові поля у product backlog, в основному для того, щоб допомогти product owner'у визначитися з його пріоритетами.

Категорія (track). Наприклад, «панель управління» чи «оптимізація». За допомогою цього поля product owner може легко вибрати усі пункти категорії «оптимізація» і задати їм низький пріоритет.

Компоненти (components) – указує, які компоненти (наприклад, база даних, сервер, клієнт) будуть зачеплені при реалізації історії. Дане поле складається з групи checkbox'ів, які відмічаються, якщо відповідні компоненти потребують змін.

Ініціатор запиту (requestor). Product owner може захотіти зберігати інформацію про усіх замовників, зацікавлених у даній задачі. Це потрібно для того, щоб тримати їх у курсі діла про хід виконання робіт.

ID у системі обліку помилок (bug tracking ID) – якщо ви використовуєте окрему систему обліку помилок, тоді у описі історії корисно зберігати посилання на всі дефекти, які до неї відносяться.

Sprint backlog – містить функціональність, обрану Product Owner із Product Backlog. Всі функції розбиті по задачах, кожна з яких оцінюється командою. Кожен день команда оцінює об'єм роботи, який необхідно провести для завершення задачі.

Burndown chart – показує, скільки вже виконано і скільки ще залишається зробити.

Планування спринта (Sprint Planning Meeting).

Проходить на початку нової ітерації Спринта:

– Із Product Backlog обираються задачі, зобов'язання по виконанню яких за спринт приймає на себе команда;

– На основі обраних задач створюється Sprint Backlog. Кожна задача оцінюється у ідеальних людино–годинах;

– Рішення задачі не повинно займати більше 12 годин або одного дня. При необхідності задача розбивається на підзадачі;

– Обговорюється та визначається, яким чином буде реалізовано цей об'єм робіт;

– Тривалість наради обмежена зверху 4–8 годинами в залежності від тривалості ітерації, досвіду команди тощо;

– (перша частина наради) Беруть участь Product Owner + Команда: обирають задачі із Product Backlog;

– (друга частина наради) Бере участь лише команда: обговорюють технічні деталі реалізації, наповнюють Sprint Backlog.

Щоденна нарада (Daily Scrum Meeting).

Відбувається кожен день протягом спринта. Є «пульсом» ходу спринта. Нараді властиві наступні обмеження:

– починається точно вчасно;

– всі можуть спостерігати, але говорять тільки обрані;

– триває не більш ніж 15 хвилин;

– проводиться в одному і тому ж місці протягом одного спринта.

Протягом наради кожен член команди відповідає на 3 запитання:

– Що зроблено з моменту попередньої щоденної наради?;

– Що буде зроблено з моменту поточної наради до наступної?;

– Які проблеми заважають досягненню цілей спринта? (Над рішенням цих проблем працює ScrumMaster. Зазвичай це рішення проходить за рамками щоденної наради і у складі осіб, що безпосередньо займаються даною перешкодою.)

Демонстрація (Sprint Review Meeting):

– Проходить у кінці ітерації (спринта).

– Команда демонструє внесок функціональності до продукту всім зацікавленим особам.

– Залучається максимальна кількість глядачів.

– Усі члени команди беруть участь у демонстрації (одна людина на демонстрацію або кожен показує, що зробив за спринт).

– Обмежена 4–ма годинами в залежності від тривалості ітерації і змін у продукті.

Ретроспектива (Sprint Retrospective):

– Члени команди висловлюють свою думку про минулий спринт.

– Відповідають на два основних запитання: Що було зроблено добре у минулому спринті?; Що потрібно покращити в наступному?.

– Виконують покращення процесу розробки (вирішують питання та фіксують вдалі рішення).

– Обмежена 1–3ма годинами.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою RC6 – симетричний блоковий криптографічний алгоритм, похідний від алгоритму RC5. Був створений Роном Рівестом, Меттом Робшау і Реєм Сіднеєм для задоволення вимог конкурсу Advanced Encryption Standard (AES). Алгоритм був одним з п'яти фіналістів конкурсу, був також представлений NESSIE і CRYPTREC. Є власницьким (пропрієтарним) алгоритмом, і запатентований RSA Security, одначе дія патентів сплила, і зараз алгоритм знаходиться у відкритому доступі. В той же час, "RC6" залишається зареєстрованою торговою маркою RSA.

Варіант шифру RC6, заявлений на конкурс AES, підтримує блоки довжиною 128 біт і ключі довжиною 128, 192 і 256 біт, але сам алгоритм, як і RC5, може бути налаштований для підтримки більш широкого діапазону довжин як блоків, так і ключів (від 0 до 2040 біт)[1]. RC6 дуже схожий на RC5 за своєю структурою і також досить простий у реалізації.

Є фіналістом AES, проте одна з примітивних операцій – операція множення, повільно виконується на певному обладнанні і ускладнює реалізацію шифру на ряді апаратних платформ і, що виявилося сюрпризом для авторів, на системах з архітектурою Intel IA-64 також реалізована досить погано. В даному випадку алгоритм втрачає одну зі своїх ключових переваг – високу швидкість

виконання, що стало причиною для критики і однією з перепон для обрання як нового стандарту.

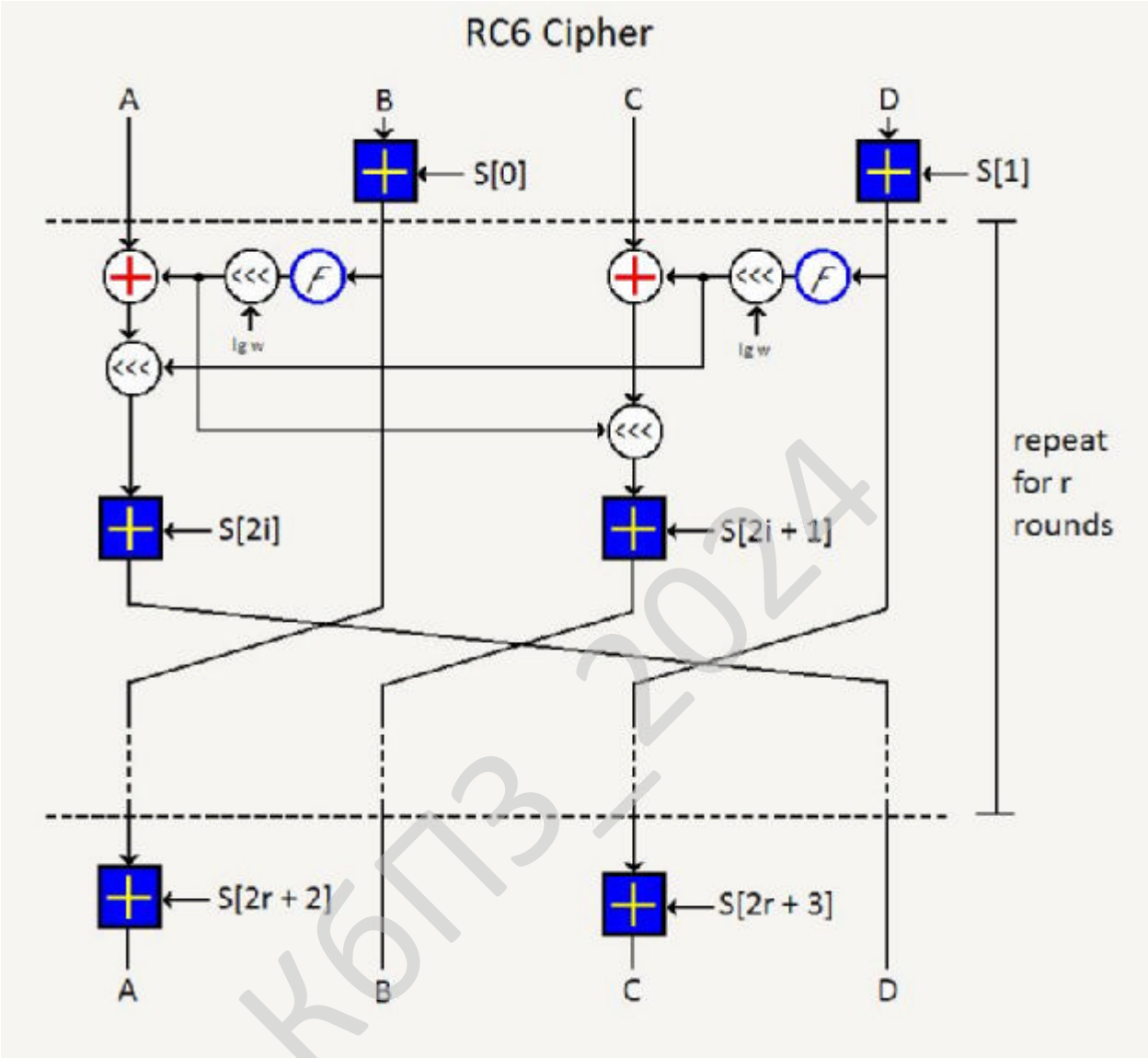

Рисунок 4.3 – Структура RC6

Деталі RC6

Так само, як і RC5, RC6 – повністю параметризована сім'я алгоритмів шифрування. Для специфікації алгоритму з конкретними параметрами, прийнято позначення RC6-w/r/b, де

– W – довжина машинного слова в бітах.

– R – число раундів.

– B – довжина ключа в байтах. Можливі значення 0 .. 255 байт.

Для того щоб відповідати вимогам AES, блочний шифр повинен працювати з 128-бітовими блоками. Так як RC5 – виключно швидкий блочний шифр, розширення його, щоб працювати з 128-бітовими блоками, привело б до використання двох 64-бітових робочих регістрів. Але архітектура і мови програмування ще не підтримують 64-бітні операції, тому довелося змінити проект так, щоб використовувати чотири 32-бітних регістри замість двох 64-бітних.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ системи безперервної роботи додатків при серверній віртуалізації яке зображено на рисунку 5.1. Обчислення виконується через консольний інструмент з подальшою передачею результатів до інтерфейсу. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Блоку виведення знайдених помилок.

– Розділу виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

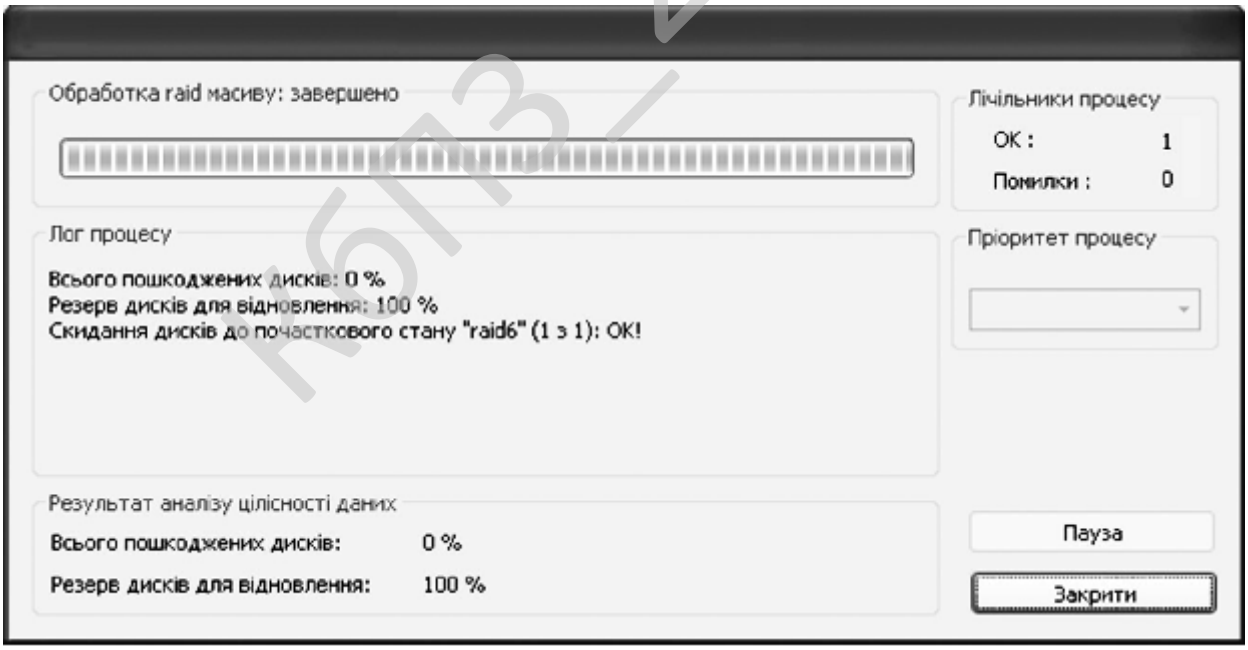

Рисунок 5.1 – Головне вікно ПЗ

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

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

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

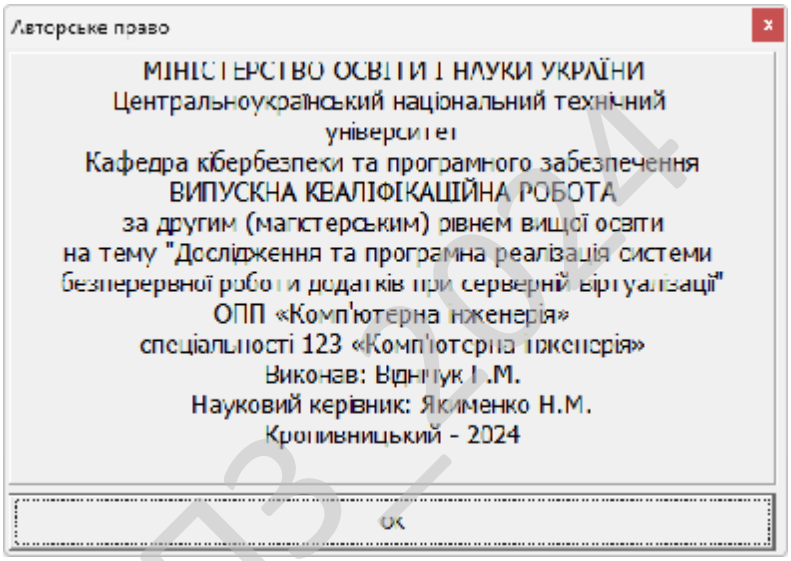

Рисунок 5.2 – Авторське право

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

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

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

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

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

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

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

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

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

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

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

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

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

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

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

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

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

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

– У програмі можуть бути пропущені деякі маршрути.

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

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

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

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

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

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

Обрано умови розповсюдження – commercial software.

Програмне забезпечення, створене комерційною організацією з метою отримання прибутку від його використання іншими, наприклад, шляхом продажу копій.

Найважливішою особливістю комерційний програмних продуктів є підтримка великих компаній, прямо зацікавлених у поширенні програм. Багато організацій надають виключно платну підтримку своїх продуктів, такий підхід, як правило, використовують організації надають відкриті вихідні коди. Для продуктів, що розповсюджуються на комерційній основі діють зазвичай безкоштовні служби підтримки, покликані збільшити рівень довіри у клієнтів і потенційних покупців.

Далеко не завжди, але як правило терміни критично важливих змін в комерційних продуктах значно менше, ніж у некомерційних проектів.

Це пов'язано з тим, що над комерційним продуктом працюють цілі групи розробників і ця робота є їх основним заняттям. Розробникам-початківцям як правило доводиться шукати додаткові способи заробітку, і це збільшує час, що витрачається на доповнення і зміни програм. Так як основним рушійним фактором створення комерційного ПЗ є одержання прибутку, то комерційні програмні продукти першими заповнюють вільні ніші та пропонують варіанти вирішення завдань відразу по мірі виявлення вакууму в будь-якому секторі ринку.

Окремий вид комерційних програм, коли їх розробка оплачується безпосередньо замовником. Такі програми найчастіше позбавлені всіх переваг комерційних продуктів, оскільки мають обмежений бюджет, але більш адаптовані до вимог замовника, ніж аналоги.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи безперервної роботи додатків при серверній віртуалізації.

Метою розробки є дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації.

Об'єктом дослідження є процес безперервної роботи додатків при серверній віртуалізації.

Предметом дослідження є методи безперервної роботи додатків при серверній віртуалізації.

Методи дослідження базуються на методах теорії комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод безперервної роботи додатків при серверній віртуалізації.

– Розроблено вітчизняний продукт безперервної роботи додатків при серверній віртуалізації, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації можуть зацікавити кілька категорій спеціалістів і компаній (рисунок 7.1).

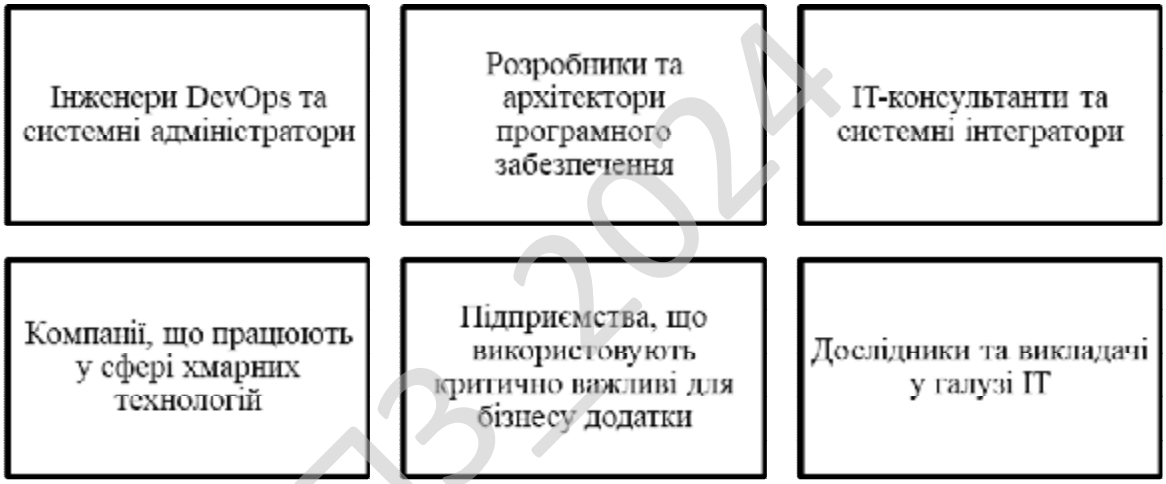

Рисунок 7.1 – Цільова аудиторія

Для інженерів DevOps та системних адміністраторів це допоможе вдосконалити стратегії забезпечення високої доступності додатків, підвищити ефективність серверної віртуалізації та оптимізувати ресурси.

Розробники та архітектори програмного забезпечення зможуть використати результати для вдосконалення інфраструктури своїх додатків, забезпечуючи безперервність роботи та стійкість до відмов.

ІТ-консультанти та системні інтегратори можуть пропонувати нові рішення для своїх клієнтів, використовуючи дослідження як частину аргументованих рекомендацій щодо вибору підходів до серверної віртуалізації.

Оскільки багато хмарних провайдерів використовують серверну віртуалізацію для забезпечення доступності своїх послуг, то результати досліджень та розробок можуть допомогти вдосконалити стратегії відмовостійкості.

Такі організації, як банки, державні структури, медичні установи та великі корпорації, зацікавлені в мінімізації ризиків простоїв, тож зможуть інтегрувати ці напрацювання в свою IT-інфраструктуру.

Для дослідників та викладачів у галузі IT результати можуть стати основою для нових досліджень та освітніх матеріалів, які допоможуть студентам і фахівцям зрозуміти сучасні підходи до забезпечення безперервності роботи додатків. Такий інтерес може стимулювати подальший розвиток рішень, спрямованих на підвищення стабільності та ефективності серверної віртуалізації.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для оцінки привабливості програмної реалізації системи безперервної роботи додатків при серверній віртуалізації за допомогою експертних методів можна використовувати метод експертного опитування з бальною оцінкою. Такий підхід дозволяє зібрати думки кількох експертів, оцінити різні аспекти системи, а також узагальнити їх для отримання зваженого результату. Спочатку формуємо критерії, за якими буде оцінюватись програмне рішення (рисунок 7.2). Обираємо експертів, які є фахівцями в серверній віртуалізації, забезпеченні безперервності роботи додатків, розробці хмарних рішень або мають досвід в IT-інфраструктурі. Експерти оцінюють кожен критерій за визначеною шкалою, наприклад, від 1 до 10, де 1 – найнижчий рівень привабливості, а 10 – найвищий. Для кожного критерію підраховуємо середню оцінку від усіх експертів (таблиця 7.1). Якщо потрібно, можна також зважити оцінки експертів, враховуючи їхній досвід чи релевантність знань.

Стійкість до відмов (наскільки система захищена від збоїв).

Продуктивність (ефективність використання ресурсів).

Масштабованість (легкість адаптації системи до більших навантажень).

Легкість обслуговування (вимоги до підтримки та оновлення).

Вартість реалізації (витрати на розробку, впровадження і підтримку).

Сумісність (здатність працювати з різними платформами або системами).

Рисунок 7.2 – Критерії експертної оцінки

Таблиця 7.1 – Зведені результати експертних оцінок

Критерій	Оцінка експерт 1	Оцінка експерт 2	Оцінка експерт 3	Середня оцінка
Стійкість до відмов	8	9	8	8.33
Продуктивність	7	8	7	7.33
Масштабованість	9	8	8	8.33
Легкість обслуговування	6	7	7	6.67
Вартість реалізації	8	7	6	7.00
Сумісність	7	8	7	7.33

Розраховуємо загальний індекс привабливості програмного рішення як середньозважене значення всіх критеріїв.

Середнє значення всіх критеріїв:

$$(8.33 + 7.33 + 8.33 + 6.67 + 7.00 + 7.33) / 6 \approx 7.50$$

Індекс привабливості, отриманий в результаті експертної оцінки, складає 7.50. Це свідчить про те, що запропоноване рішення є досить привабливим і відповідає ключовим вимогам, хоча має потенційні зони для вдосконалення, наприклад, у легкості обслуговування.

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості програмної реалізації системи безперервної роботи додатків при серверній віртуалізації рекомендується використовувати метод методу оцінки вартості на основі діяльності (Activity-Based Costing, ABC). Цей метод дозволяє точно оцінити вартість, враховуючи всі ключові етапи та процеси, які пов'язані з розробкою, реалізацією та підтримкою системи.

Чому метод Activity-Based Costing (ABC) підходить для цього проекту: метод ABC дозволяє детально розрахувати вартість кожного процесу або компонента системи, наприклад, вартість розробки окремих модулів, тестування, впровадження та підтримки; у системах серверної віртуалізації є суттєві непрямі витрати (наприклад, енергія, оренда серверів або зарплата команди підтримки). Метод ABC допомагає точно розподілити ці витрати на відповідні процеси; ABC враховує витрати як на програмну розробку, так і на інфраструктуру та необхідні ресурси (сервери, мережеве обладнання тощо), що є важливим для створення надійної системи безперервної роботи; такий метод також враховує витрати, пов'язані з майбутнім обслуговуванням, що дозволяє оцінити загальну вартість володіння (Total Cost of Ownership, TCO), включаючи витрати на підтримку та оновлення.

Якщо є обмеження у ресурсах або бракує доступу до деталей усіх процесів, можна також використовувати метод оцінки за аналогами (Comparative or Analogous Estimation), порівнюючи вартість з аналогічними проектами, або метод експертної оцінки. Однак вони можуть мати нижчу точність.

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

Економічна ефективність від впровадження системи безперервної роботи додатків при серверній віртуалізації може включати кілька складових. Розглянемо приклад оцінки на основі основних факторів економії та вигод, що виникають завдяки безперервній роботі додатків, мінімізації простоїв і оптимізації витрат на інфраструктуру.

Впровадження системи безперервної роботи додатків при серверній віртуалізації дозволяє заощадити 75 000 доларів на рік завдяки зниженню втрат від простоїв, економії на технічній підтримці, оптимізації витрат на інфраструктуру та підвищенню продуктивності.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Для просування проєкту програмної реалізації системи безперервної роботи додатків при серверній віртуалізації можна застосувати багатокроковий алгоритм, що охоплює як початкове планування, так і активне просування через маркетингові канали (рисунок 7.3).

Таке просування допоможе ефективно донести переваги системи до цільової аудиторії, підвищити обізнаність про проєкт і збільшити кількість клієнтів.

Таблиця 7.2 – Основні показники впровадження проєкту

Вхідні дані: **Зниження втрат від простоїв**: До впровадження система мала середній час простою 10 годин на рік, що призводило до втрати 50 000 доларів щорічно. **Економія на технічній підтримці**: Завдяки впровадженню системи безперервної роботи вдалося зменшити необхідність у позапланових роботах та знизити витрати на підтримку з 30 000 до 20 000 доларів на рік. **Зниження витрат на інфраструктуру**: Віртуалізація дозволяє зменшити потребу в фізичному обладнанні, скоротивши витрати на 15%. **Підвищення продуктивності**: За рахунок безперервної роботи зростає ефективність використання ресурсів та зменшується необхідність в розширенні штату на 5%.
Розрахунок економічної ефективності: **Зниження втрат від простоїв**: Зменшення часу простою з 10 до 2 годин на рік. Потенційна економія: 50000×810=40000 доларів на рік. **Економія на технічній підтримці**: Зниження витрат з 30 000 до 20 000 доларів на рік. Потенційна економія: 30000−20000=10000 доларів на рік. **Зниження витрат на інфраструктуру**: Витрати на фізичне обладнання до впровадження становили 100 000 доларів на рік. Економія: 100000×0.15=15000 доларів на рік. **Підвищення продуктивності**: Зменшення необхідності у додаткових ресурсах дозволяє зекономити 5% від загальних витрат на робочу силу, що, наприклад, складає 200 000 доларів на рік. Потенційна економія: 200000×0.05=10000 доларів на рік. Загальна економічна ефективність: Сума всіх зекономлених коштів на рік: 40000+10000+15000+10000=75000 доларів на рік

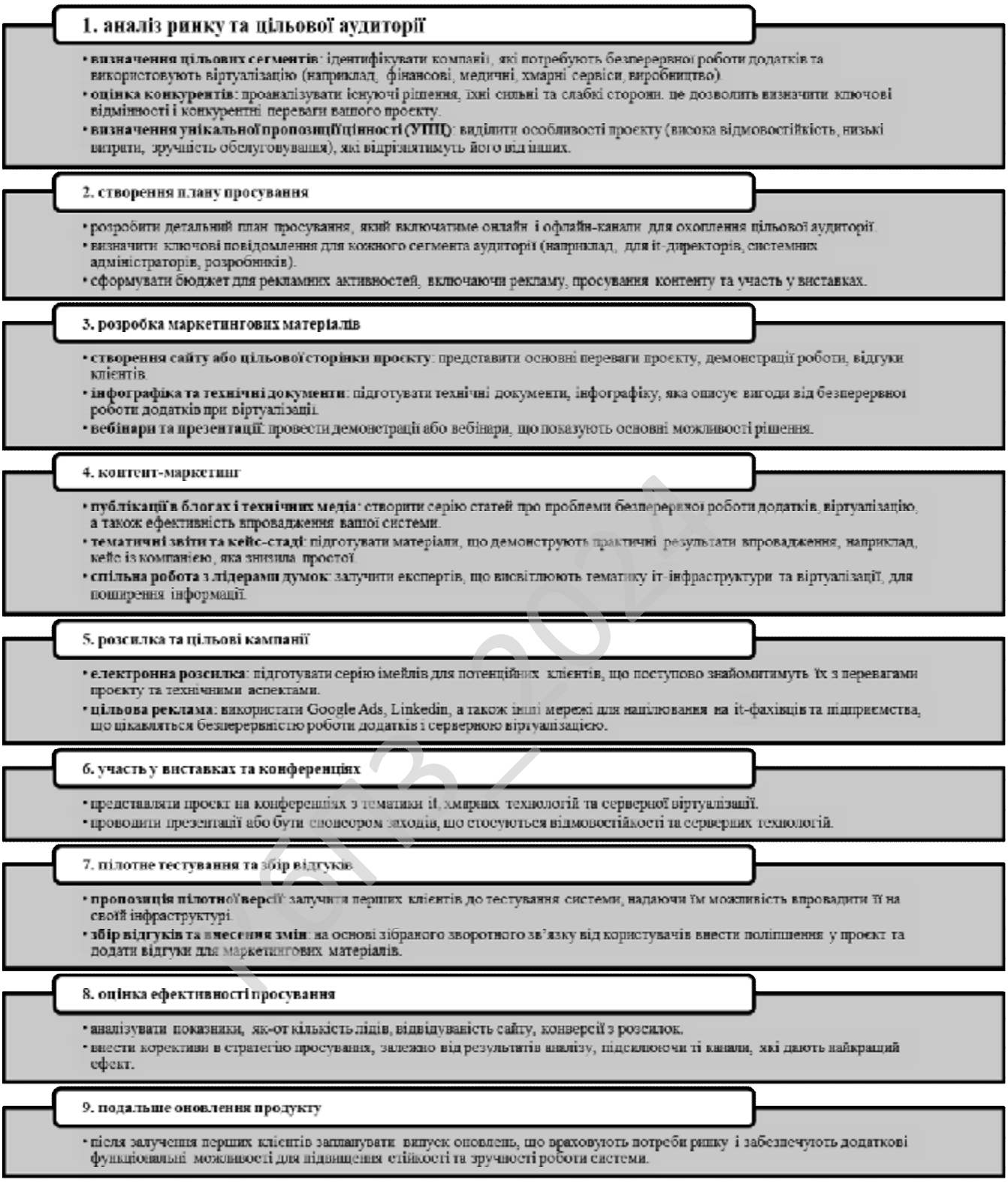

Рисунок 7.3 – Алгоритм просування проєкту

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту та шляхів реалізації такого проєкту можна сфокусуватися на розробці цільових каналів продажу та гнучких стратегій реалізації:

– прямий продаж через IT-консалтинг: співпрацювати з компаніями, що надають консалтингові послуги у сфері віртуалізації та хмарних рішень, щоб вони могли рекомендувати вашу систему своїм клієнтам;

– партнерство з провайдерами хмарних сервісів: створення спільних пропозицій з провайдерами, що пропонують хмарні послуги. такі партнери можуть рекомендувати вашу систему безперервної роботи своїм користувачам як рішення для забезпечення відмово стійкості;

– інтеграція з дистриб'юторами програмного забезпечення: партнерство з компаніями, що займаються дистрибуцією програмного забезпечення для серверної віртуалізації (наприклад, VMware, Hyper-V, Citrix), може допомогти значно розширити географію збуту;

– пропозиція підписки на SaaS: запровадження моделі «програмного забезпечення як послуги» (SaaS) дозволить клієнтам користуватися системою безперервної роботи за підпискою, що знижує вартість входу і сприяє ширшому розповсюдженню.

– оновлення на основі зворотного зв'язку: SaaS-модель також дозволяє постійно оновлювати систему відповідно до потреб клієнтів, що підвищує лояльність і утримує клієнтів;

– співпраця з реселерами: залучити реселерів, які вже мають клієнтські бази серед компаній, зацікавлених у надійних системах віртуалізації. Це дозволяє охопити ринки, на яких ваша команда не представлена напряму.

– програма для партнерів: запропонувати реселерам вигідні умови продажу та інструктаж для демонстрації продукту. Це сприятиме збільшенню кількості партнерів і заохотить їх до просування вашого рішення;

– технічні демо та підтримка впровадження: надання безкоштовних демонстрацій або пілотних проєктів для великих клієнтів допоможе їм оцінити переваги безперервної роботи віртуалізованих додатків;

– цілодобова технічна підтримка: пропозиція цілодобової підтримки підвищить цінність вашої системи для клієнтів, які залежать від безперервної роботи додатків, забезпечуючи швидке реагування на будь-які проблеми;

– таргетована реклама для IT-рішень: використання Google Ads, LinkedIn Ads та інших платформ з націленням на IT-директорів, системних адміністраторів та розробників, що потребують рішень для віртуалізації;

– просування через контент-маркетинг: створення статей, технічних блогів, вебінарів, які розкривають вигоди системи для забезпечення відмовостійкості, та поширення їх через канали, що читають IT-фахівці;

– SEO для релевантних пошукових запитів: Оптимізація сайту для запитів, пов'язаних із безперервною роботою додатків та серверною віртуалізацією, допоможе залучити більше потенційних клієнтів;

– розрахунок ROI для потенційних клієнтів: пропозиція попереднього розрахунку окупності інвестицій (ROI) допоможе клієнтам зрозуміти фінансову вигоду від впровадження вашої системи;

– кейси та відгуки клієнтів: публікація історій успіху та відгуків допоможе продемонструвати економічну ефективність і надійність вашого рішення, сприяючи його популяризації;

– такі кроки сприятимуть максимізації охоплення ринку, а також залученню й утриманню клієнтів, що прагнуть забезпечити безперервну роботу своїх додатків у віртуалізованих середовищах.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключові фактори успіху для проєкту програмної реалізації системи безперервної роботи додатків при серверній віртуалізації включають технічні, організаційні та ринкові аспекти (рис. 7.4).

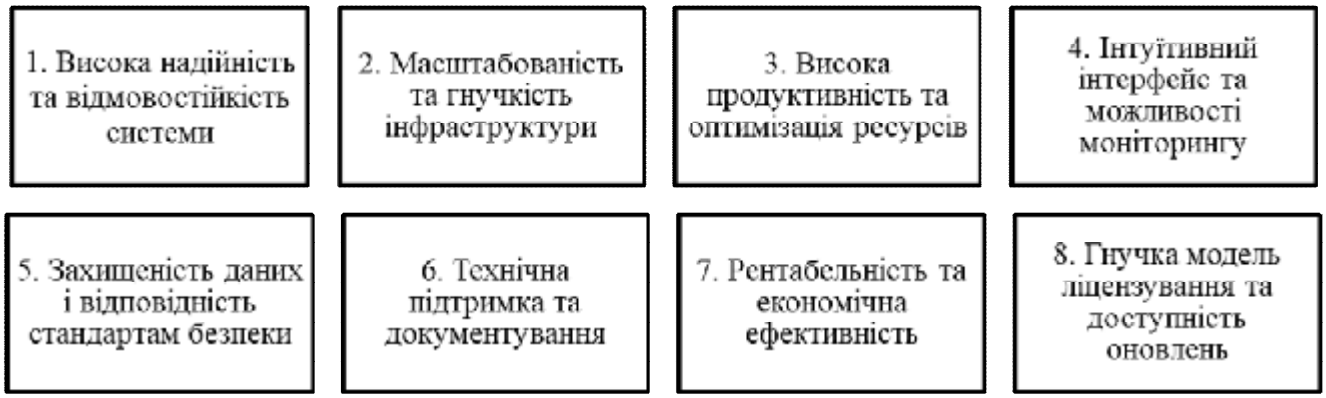

Рисунок 7.4 – Ключові фактори успіху проєкту

Ці фактори є критично важливими для створення успішного продукту, який зможе витримати конкурентний тиск і здобути довіру клієнтів завдяки високій якості, надійності та економічній вигоді.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Сучасний розвиток технічного та технологічного стану виробництва передбачає постійну автоматизацію та оптимізацію виробничих процесів. Комп’ютер – невід’ємна складова сучасного життя. За допомогою обчислювальної техніки вирішують складні робочі задачі, ведуться наукові дослідження, створюються архітектурні креслення і твори мистецтва. Сьогодні, напевно, важко уявити компанію, господарська діяльність в якій здійснювалась би без використання комп’ютерної техніки. Незважаючи на видиму безпеку та розвитку сучасних технологій, при роботі за комп'ютером є ряд чинників, які можу вплинути на здоров'я людини. Через масовий характер робіт, що виконуються працівниками за допомогою комп’ютера, законодавством України чітко врегульовано норми та вимоги до використання комп’ютерної техніки на підприємстві, безпосередньо й охорона праці на підприємстві при роботі за комп’ютером.

Законом України “Про охорону праці” [1] регламентуються загальні положення державної політики в галузі охорони праці, а реалізуються ці положення, зокрема, Вимогами щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, затверджені наказом Мінсоцполітики від 14.02.2018р. № 207, зареєстровані в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 [2].

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи і достатньо великим навантаженням на м'язи рук при роботі з клавіатурою ЕОМ.

У розділі даної магістерської роботи висвітлюються основні питання охорони праці працівників, робота яких пов’язана з роботою за комп’ютером, планування робочого приміщення, де працюють користувачі ПК; параметри мікроклімату, освітленість робочих місць та виробничих приміщень; шумові завади.

Правильна організація і раціональне устаткування робочого місця можливість ефективно і з якнайменшими витратами праці виконувати свої функції, плідно спілкуватися співробітниками і підлеглими, підтримувати високу працездатність і робочий настрій. Велике значення має раціональна конструкція і розташовує елементів робочого місця, що важливе для підтримки оптимальної робочої пози людини-оператора, а також необхідно дотримувати правильний режим праці і відпочинку.

Що стосується питання охорони праці людини необхідно вирішувати на всіх стадіях трудового процесу незалежно від виду професійної діяльності.

Забезпечення безпечних і здорових умов праці в значній мірі залежить від правильної оцінки небезпечних, шкідливих виробничих факторів. Однакові по складності зміни в організмі людини можуть бути викликані різними причинами. Це можуть бути фактори виробничого середовища, надмірне фізичне і розумове навантаження, нервово-емоційна напруга, а також різне сполучення цих причин.

Робота працівників пов’язана з роботою за комп’ютером, тому актуальною є розгляд саме умов праці та стану охорони праці працівників які постійно працюють з комп’ютерною технікою.

Завдання даного розділу полягає у тому, щоб розробити якісний програмний продукт необхідно організувати безпеку на робочому місці програміста. Під час проектування безпеки робочому місці з ПК необхідно домагатися високої якості та надійності технічного забезпечення, але й створювати комфортні параметри довкілля для розробників.

8.2 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

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

Таблиця 8.1 – Розміри приміщення

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

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

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

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

У зазначеному приміщенні працюють 4 людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об’єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2], але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [5] та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-

обчислювальних машин»). Таним чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

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

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Вологість,%	Швидкість повітря, м/с	Температура,°С	Вологість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-23	40-55	0,1
Тепла	23-25	50-70	0,1	24-25	50-65	0,11

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

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

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

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

З 2019 року діють Державні будівельні норми україни “Природне і штучне освітлення” – ДБН В.2.5-28:2018 [1], у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних.

Працю працівника, який постійно працює за комп’ютером, згідно ДБН В.2.5-28:2018 [1], можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В).

Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк. [1],

Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.3 Розробка заходів з умов поліпшення охорони праці

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

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

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значеннь;

Таким чином можна припустити, що основною причиною можнивого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог. Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга).

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

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

Для підвищення ефективності системи управління охорони праці дуже важлива роль належить формуванню і розвитку інформаційної культури фахівців IT-технологій

8.4 Розрахункова частина

Для захисного штучного заземлення застосовуються вертикальні електроди: металевий куток 45х45х5 мм, довжиною L=2.5 м, та горизонтальний електрод – металева полоса з перетином 40х4 мм. Напруга – 220/380 В. Розрахункова схема розташування заземлюючих електродів – у ряд (рис. 8.1).

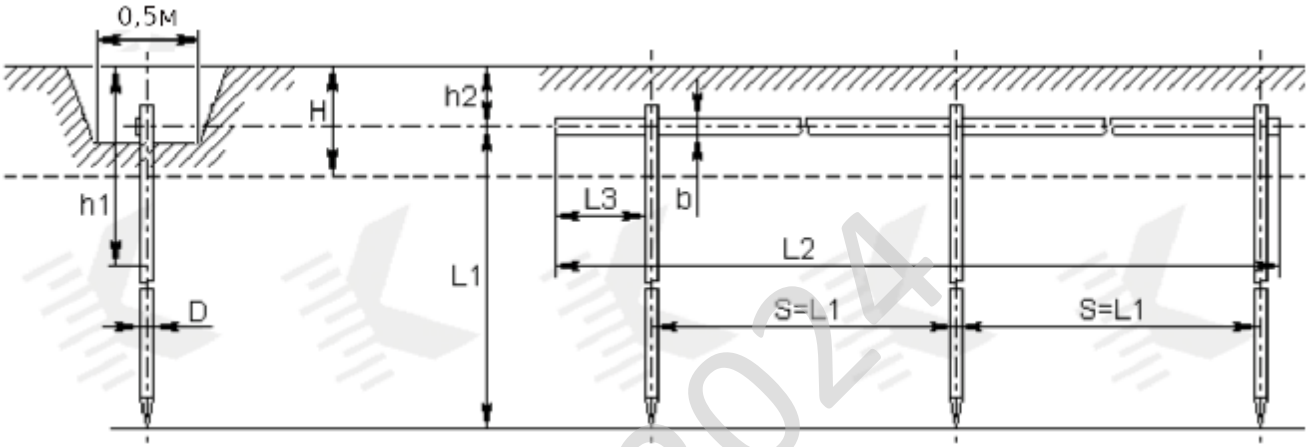

Рисунок 8.1 – Схема штучного заземлення.

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення:

– тип верхнього шару грунта – чорнозем, нижнього шару грунта – глина (питомий опір ρ_2 = 40 Ом·м);

– умовна товщина верхнього шару грунта: H=0.5 м;

– відстань між вертикальними заземлювачами (електродами) A=3 м;

– глибина закладення горизонтального контура заземлення t=0.8 м;

– опір заземлювача, який нормується: $R_{ЗН}$ = 4 Ом.

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

Виконаємо розрахунок захисного заземлення.

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,8+2.5/2=2.05 \text{ м.}$$

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

$$\rho = \psi \cdot \rho_2 = 1.36 \cdot 40 = 54.4 \text{ Ом·м.}$$

де $\psi = 1.36$ – табличне значення коефіцієнта сезонності для відповідної климатичной зони у багатошаровому грунті [6];

$\rho = 40$ Ом·м. – табличне значення питомого опору нижнього шару грунту (глина) [11].

Еквівалентний діаметр вертикального електрода (кутка) [11]:

$$D_в=0.95 \cdot К= 0.95 \cdot 45 = 0.043 \text{ м.}$$

де К= 45 мм – розмір металевого кутка (заданий).

Відношення A/L=3/2.5=1.2.

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

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

$$=0.366 \cdot (54.5/2.5) \cdot [(\lg(2 \cdot 2.5/0.0475)+(1/2) \cdot \lg((4 \cdot 2.3+2.5)/(4 \cdot 2.3-2.5))]=17.55 \text{ Ом.}$$

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

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

$$N=R_О /(К_{ев} \cdot R_{ЗН})= 17.55 / (0.8 \cdot 4)=5.48 \approx 6 \text{ шт.}$$

Визначаємо довжину з'єднуючої полоси:

$$L_П =1.05 \cdot A \cdot N = 1.05 \cdot 2.5 \cdot 5.48 = 17.28 \approx 18 \text{ м.}$$

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

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

$$=0.366 \cdot (40 \cdot 5/17.28) \cdot \lg((2 \cdot 17.28^2)/(0.04 \cdot 0.8))=18 \text{ Ом.}$$

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

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

Загальний опір розтіканню електричного струму заземлювача [11]:

$$R = (R_О \cdot R_П)/(R_О \cdot \eta_П + N \cdot R_П \cdot Кев)=$$

$$=(17.55 \cdot 18)/(17.55 \cdot 0.75 + 5.48 \cdot 18 \cdot 0.8) = 3.43 \text{ Ом.}$$

де $\eta_П$ = 0.75 – табличне значення коефіцієнта екранування з'єднуючої полоси [11].

Умова $R \leq R_{ЗН}$ виконується, $3.43 \leq 4$.

При необхідності можна зменшити кількість електродів заземлювача зменшивши загальний опір розтіканню електричного струму заземлювача методом зменшення питомого опору ґрунта, домішуючи у грунт безпосередньо навколи електродів заземлювача розчини солей NaCl, CaCl, сажу, соду, шлак, коксову дрібницю, або спеціальні суміші.

8.5 Висновки до розділу

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

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

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи безперервної роботи додатків при серверній віртуалізації.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів безперервної роботи додатків при серверній віртуалізації.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем безперервної роботи додатків при серверній віртуалізації.

– Досліджена система безперервної роботи додатків при серверній віртуалізації.

– На основі отриманих результатів досліджень створена програмна реалізація системи безперервної роботи додатків при серверній віртуалізації.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання безперервної роботи додатків при серверній віртуалізації.

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

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

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

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм RC6.

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

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Віднічук Г.М. Дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2024.

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

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

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

10. 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.

11. 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.

12. 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.

13. 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.

14. 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.

15. 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.

16. 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.

17. 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.

18. 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.

19. 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.

20. 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.

21. 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.

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

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

28. 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.

29. 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.

30. 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.

31. 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.

32. 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

33. 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.

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. 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).

40. 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

41. 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.

42. 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.

43. 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.

44. 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.

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

46. 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.

47. 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.

48. 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.

49. Смірнов О.А., Смірнов С.А., Дідик А.К. Метод безпечної маршрутизації метаданих у хмарні антивірусні системи. Системи озброєння та військова техніка. – Випуск 2 (46) – Х.: ХУПС – 2016. – С. 146-149.

50. Смірнов О.А., Кавун С.В., Доренський О.П., Вялкова В.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 151 с.

51. Смірнов О.А., Кавун С.В., Коваленко О.В., Дрєєв О.М. Мережні інформаційні технології. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 159 с.

52. Смірнов О.А., Кавун С.В., Коваленко О.В., Доренський О.П., Дрєєв О.М., Вялкова В.І. Комп'ютерні мережі. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 233 с.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи безперервної роботи додатків при серверній віртуалізації.

2 Підстава для розробки

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

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

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

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи безперервної роботи додатків при серверній віртуалізації;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Visual C#.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Економічні вимоги

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

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

8 Вимоги щодо охорони праці

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

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 111 аркушів.

10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Маркетингове та економічне обґрунтування ІТ-проєкту.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

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

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

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

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

_____ Якименко Н.М.

Дослідження та програмна реалізація системи безперервної роботи додатків при серверній віртуалізації

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 72

Літера: РП

Кропивницький – 2024 року

Файл RSRaidBase.cs – робота з RAID масивами для забезпечення системи безперервної роботи додатків при серверній віртуалізації

```
using System;
using System.Threading;

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

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

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

        #endregion Delegates

        #region Public Properties & Data

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

                } else
                {
                    return false;
                }
            }
        }

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

```
            return this.configIsOK;

        } else
        {
            return false;
        }
    }
}

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

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

        } else
        {
            return false;
        }
    }
}

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

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

        } else
        {
            return -1;
        }
    }
}

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

/// <summary>
/// Кількість томів для відновлення
/// </summary>
public int EccCount
{
    get
    {
```

```
                if (!InProcessing)
                {
                    return this.m;

                } else
                {
                    return -1;
                }
            }
        }

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

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

                } else
                {
                    return -1;
                }
            }
        }

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

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

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

                            break;
                        }
```

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

                            break;
                        }

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

                            break;
                        }

                        case 3:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.AboveNormal;

                            break;
                        }

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

                            break;
                        }
                    }

                    // Установлюємо обраний пріоритет процесу
                    this.thrRSMatrixForming.Priority = this.threadPriority;
                }
            }
        }

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

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

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

        #endregion Public Properties & Data

        #region Data

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

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

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

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

        // Основна конфігурація змінилася?
        protected bool mainConfigChanged;

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

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

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

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

        #endregion Data

        #region Construction & Destruction

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

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

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

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

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

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
```

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

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

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

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

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

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

                // Повертаємо результат обробки
                return this.configIsOK;
            }

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

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

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

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

            return true;
        }

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

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

        /// <summary>
```

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

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

        #endregion Public Operations

        #region Protected Operations

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

            if (n > m)
            {
                maxVal = n;

            } else
            {
                maxVal = m;
            }

            double divider = maxVal / 100.0;

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

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

            return -1;
        }

        /// <summary>
```

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

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

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

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

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

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

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

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

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

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

```
            {
                progressMod1 = 1;
            }

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

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

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

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

                    return false;
                }

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

                int k_n = k * this.n;

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

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

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

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

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

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

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

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

                // Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((k % progressMod1) == 0)
                     &&
                        (OnUpdateRSMatrixFormingProgress != null)
                   )
                {
                    //...виводимо дані
                    OnUpdateRSMatrixFormingProgress(((double)(k + 1) /
(double)this.n) * percOfFirstStage);
                }

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

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Вказуємо, що декодер не сконфігурований коректно
                    this.configIsOK = false;

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

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

                    return false;
                }
            }

            return true;
        }

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

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

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

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

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

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

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

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

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

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

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

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

```
                    )
                {
                    //...виводимо дані
                    OnUpdateRSMatrixFormingProgress(((double)(i + 1) /
(double)this.m) * percOfFirstStage);
                }

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

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Вказуємо, що декодер не сконфігуровано коректно
                    this.configIsOK = false;

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

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

                    return false;
                }
            }

            return true;
        }

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

        #endregion Protected Operations
    }
}
```

Файл RSRaidEncoder.cs - кодування алгоритмом Ріда-Соломона

```
using System;
using System.Threading;

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

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

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

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

```
            int maxVolCount;

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

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

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

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

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

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

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

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

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

                this.configIsOK = true;

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

```
            return this.configIsOK;
        }

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

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

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

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

                ecc[i] = mulSum;
            }

            return true;
        }

        #endregion Public Operations

        #region Private Operations

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

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

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

                        return;
                    }

                } else
```

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

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

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

                        return;
                    }
                }

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

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

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

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

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

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

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

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

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

                        return;
                    }
                }

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

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

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

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

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

        #endregion Private Operations
    }
}
```

Файл RSRaidDecoder.cs - декодування алгоритмомом Ріда-Соломона

```
using System;
using System.Threading;

namespace RecoveryDisk
{
    /// <summary>
    /// Клас RAID-подібного декодера Ріда-Соломона
    /// </summary>
    public class RSRaidDecoder : RSRaidBase
    {
        #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 RSRaidDecoder()
        {
            // 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 RSRaidDecoder(int dataCount, int eccCount, int[] volList)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList, (int)RSType.Alternative);

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

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

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

```
        #endregion Construction & Destruction

        #region Public Operations

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

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

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

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

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

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

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

                // Кількість ітерацій, що відслідковуються прогресом, на першій
стадії
                // залежить від типу використовуваної матриці
```

```
                if (this.eRSType == (int)RSType.Alternative)
                {
                    this.iterOfFirstStage = m;

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

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

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

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

                this.configIsOK = true;

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

            return this.configIsOK;
        }

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

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

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

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

                    data[i] = mulSum;

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

```
            }
        }

        return true;
    }

    #endregion Public Operations

    #region Private Operations

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

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

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

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

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

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

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

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

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

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

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

                    return false;
                }

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

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

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

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

                        int idx = i_n + j;

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

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

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

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

```
                        int idx = i_n + j;

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

                // Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((k % progressMod1) == 0)
                     &&
                        (OnUpdateRSMatrixFormingProgress != null)
                   )
                {
                    //...виводимо дані
                    OnUpdateRSMatrixFormingProgress((((double)(k + 1) /
(double)this.n) * percOfSecondStage) + percOfFirstStage);
                }

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

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

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

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

                    return false;
                }
            }

            return true;
        }

        /// <summary>
        /// Обчислення логарифмів значень інвертованої матриці
        /// <summary>
        private void LogFCalc()
        {
            // Працюємо з рядками...
            for (int i = 0; i < this.n; i++)
            {
                // Зсув у масиві до елементів і-ой рядка
                int i_n = i * this.n;

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

                    this.FLog[idx] = this.eGF16.Log(this.FLog[idx]);
                }
            }
        }

        /// <summary>
        /// Заповнення матриці "FLog" (матриці декодера) даними
        /// </summary>
        protected override void FillFLog()
        {
```

```
            // Якщо довжина вектора наявних томів менше кількості,
            // необхідного для відновлення...
            if (this.volList.Length < this.n)
            {
                //...указуємо на помилку конфігурації
                this.configIsOK = false;

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

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

                return;
            }

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

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

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

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

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

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

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

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

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

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

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

                    return;
                }
            }

            // Перевіряємо лічильники томів на помилкове дублювання
```

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

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

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

                    return;
                }
            }

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

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

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

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

                        return;
                    }

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

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

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

                        return;
                    }
                }

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

            // Для кожного загубленого основного тому шукаємо том для
відновлення
            for (int i = 0, j = 0; i < dataVolMissCount; i++)
            {
                // Рухаємося за списком томів доти, поки не знайдемо том для
```

```
                // відновлення для затикання "дірки" (основні томи мають номера
                // менше this.n (при нумерації з нуля!))
                while (this.volList[j] < this.n)
                {
                    j++;
                }

                // Зберігаємо номер тому для заміни загубленого основного тому
                eccVolToFix[i] = this.volList[j];

                j++; // j++ дозволяє перейти до наступного пошуку
            }

            // Працюємо по рядках матриці (в ідеалі, всі рядки повинні
заповнюватися
            // рядками з одиницею на головній діагоналі, що відповідає
відсутності
            // ушкоджень, але allVolCount укаже, якими є справи з наявністю
томів)
            for (int i = 0, e = 0; i < this.n; i++)
            {
                // Індекс рядка з дисперсної матриці, що буде поміщена в матрицю
кодування
                int DRowIdx;

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

                // Якщо основний том відсутній, формуємо рядок матриці
Вандермонда
                if (allVolCount[i] == 0)
                {
                    // Обчислюємо номер рядка матриці Вандермонда, яку потрібно
вставити
                    // на місце даного рядка формованої матриці "FLog"
                    DRowIdx = eccVolToFix[e++];

                    // Указуємо, що даний рядок матриці "FLog" не тривіальна
                    this.FLogRowIsTrivial[i] = false;

                } else
                {
                    // Формуємо в матриці "FLog" нульовий рядок з одиницею на
головній діагоналі
                    // (відповідає наявному основному той)
                    DRowIdx = i;

                    // Указуємо, що даний рядок матриці "FLog" тривіальна
                    this.FLogRowIsTrivial[i] = true;
                }

                // Залежно від типу декодера беремо дані з відповідного масиву
                // (у ньому втримуються як рядки матриці Вандермонда, так і
"тривіальні" рядки,
                //  утримуючі нулі і єдиний елемент "1" на головній діагоналі)
                if (this.eRSType == (int)RSType.Dispersal)
                {
                    int bs = DRowIdx * this.n;

                    // Формування рядка в матриці кодування
                    // ("тривіальні" рядки вже втримуються в матриці "D", вони
вийшли
                    // "автоматично" на попередньому етапі обробки
MakeDispersal())
                    for (int j = 0; j < this.n; j++)
                    {
                        this.FLog[i_n + j] = this.D[bs + j];
                    }
```

```
                    } else
                    {
                        // Якщо це потрібно - формуємо "тривіальну" рядок...
                        if (this.FLogRowIsTrivial[i])
                        {
                            for (int j = 0; j < this.n; j++)
                            {
                                this.FLog[i_n + j] = 0;
                            }

                            this.FLog[i_n + i] = 1;

                        } else
                        {
                            int bs = (DRowIdx - this.n) * this.n;

                            //...а, інакше, беремо рядок матриці Вандермонда
                            for (int j = 0; j < this.n; j++)
                            {
                                this.FLog[i_n + j] = this.A[bs + j];
                            }
                        }
                    }

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

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        //...указуємо на помилку конфігурації
                        this.configIsOK = false;

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

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

                        return;
                    }
                }

                // Знаходимо зворотну матрицю для "FLog"
                if (!FInv())
                {
                    // Указуємо, що кодер зконфігуровано некоректно
                    this.configIsOK = false;

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

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

                    return;
                }

                // Обчислюємо логарифми елементів інвертованої матриці
                LogFCalc();

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

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

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

        #endregion Private Operations

        #region Public Properties

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

                } else
                {
                    return null;
                }
            }
        }

        #endregion Public Properties
    }
}
```

Файл FileAnalyzer.cs - контроль цілісності даних

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

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

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

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

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

        #endregion Delegates

        #region Public Properties & Data

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

                } else
                {
                    return false;
                }
            }
        }

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

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

        } else
        {
            return false;
        }
    }
}

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

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

        } else
        {
            return false;
        }
    }
}

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

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

        } else
        {
            return null;
        }
    }
}

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

/// <summary>
/// Всі томи для відновлення коректні?
```

```
        /// </summary>
        public bool AllEccVolsOK
        {
            get
            {
                if (!InProcessing)
                {
                    return this.allEccVolsOK;

                } else
                {
                    return false;
                }
            }
        }

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

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

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

                            break;
                        }

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

                            break;
                        }

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

                            break;
                        }

                        case 3:
                        {
```

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

                            break;
                        }

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

                            break;
                        }
                    }

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

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

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

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

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

        #endregion Public Properties & Data

        #region Data

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

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

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

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

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

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

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

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

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

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

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

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

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

        #endregion Data

        #region Construction & Destruction

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

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

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

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

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

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

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

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

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

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

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

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

        #endregion Construction & Destruction

        #region Public Operations

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

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

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

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

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

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

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

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

                return false;
            }

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

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

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

                return false;
            }

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

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

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

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

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

                // Повертаємо результат обробки
                return this.processedOK;
            }

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

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

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

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

            // Повідомляємо, що все нормально
            return true;
        }

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

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

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

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

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

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

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

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

```
                return false;
            }

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

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

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

                return false;
            }

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

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

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

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

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

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

                // Повертаємо результат обробки
                return this.processedOK;
            }

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

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

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

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

```
            // Повідомляємо, що все нормально
            return true;
        }

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

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

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

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

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

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

        #endregion Public Operations

        #region Private Operations

        /// <summary>
        /// Обчислення й запис у кінець файлів значення CRC-64
        /// </summary>
        private void WriteCRC64()
        {
            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = (this.dataCount + this.eccCount) / 100;

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

            // Піддаємо обробці всі томи
            for (int volNum = 0; volNum < (this.dataCount + this.eccCount);
volNum++)
            {
                // Зчитуємо первісне ім'я файлу
                String fileName = this.fileName;
```

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

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

                // Робимо обчислення CRC-64 для кожного файлу
                if (this.eFileIntegrityCheck.StartToWriteCRC64(fileName, true))
                {
                    // Цикл очікування завершення обробки файлу
                    while (true)
                    {
                        // Якщо не виявили встановленої події "executeEvent",
                        // те користувач хоче, щоб ми поставили обробку на паузу
-
                        if (!ManualResetEvent.WaitAll(this.executeEvent, 0,
false))
                        {
                            //...припиняємо роботу контрольованого алгоритму...
                            this.eFileIntegrityCheck.Pause();

                            //...програма переходить у режим сна
                            ManualResetEvent.WaitAll(this.executeEvent);

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

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

                        //...якщо одержали сигнал до того, щоб прокинутися -
                        // переходимо на нову ітерацію, тому що прокидаємося
                        // перед постановкою на паузу...
                        if (eventIdx == 0)
                        {
                            //...попередньо скинувши подію, що змусила нас
прокинутися
                            this.wakeUpEvent[0].Reset();

                            continue;
                        }

                        //...якщо одержали сигнал до виходу з обробки...
                        if (eventIdx == 1)
                        {
                            //...зупиняємо контрольований алгоритм
                            this.eFileIntegrityCheck.Stop();

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

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

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

                            return;
                        }

                        //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
```

```
                        if (eventIdx == 2)
                        {
                            //...exitимо із циклу очікування завершення (цього й
чекали в while(true)!)
                            break;
                        }

                    } // while(true)

                } else
                {
                    // Скидаємо прапор коректності результату
                    this.processedOK = false;

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

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

                    return;
                }

                // У зв'язку із закриттям великої кількості файлових потоків
                // необхідно дочекатися запису змін, внесених потоком
                // кодування в тіло класу. Потік уже не працює, але
                // установлена їм Булевська властивість, можливо, ще
                // "не виявилося"
                for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                {
                    if (!this.eFileIntegrityCheck.Finished)
                    {
                        Thread.Sleep((int)WaitTime.MinWaitTime);

                    } else
                    {
                        break;
                    }
                }

                // Якщо цикли очікування закриття файлових потоків не привели до
бажаного
                // результату - це помилка
                if (!this.eFileIntegrityCheck.ProcessedOK)
                {
                    // Указуємо на те, що обробка не була завершена коректно
                    this.processedOK = false;

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

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

                    return;
                }

                // Виводимо прогрес обробки
                if (
                        ((volNum % progressMod1) == 0)
                     &&
                        (OnUpdateFileAnalyzeProgress != null)
                    )
                {
                    OnUpdateFileAnalyzeProgress(((double)(volNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                }
```

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

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Указуємо на те, що обробка була перервана
                    this.processedOK = false;

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

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

                    return;
                }
            }

            // Повідомляємо про закінчення процесу обробки
            if (OnFileAnalyzeFinish != null)
            {
                OnFileAnalyzeFinish();
            }

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

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

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

          /// <summary>
        /// Обчислення й перевірка значення CRC-64, записаного наприкінці файлу
        /// </summary>
        private void AnalyzeCRC64()
        {
            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = (this.dataCount + this.eccCount) / 100;

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

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

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

            // Індекс у масиві томів
            int volListIdx = 0;

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

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

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

            // Ім'я файлу для обробки
            String fileName;

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

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

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

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

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

                                    //...програма переходить у режим сна
                                    ManualResetEvent.WaitAll(this.executeEvent);

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

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

                                //...якщо одержали сигнал до того, щоб
прокинутися -
                                // переходимо на нову ітерацію, тому що
прокидаємося
                                // перед постановкою на паузу...
```

```
                                if (eventIdx == 0)
                                {
                                    //...попередньо скинувши подію, що змусила
нас прокинутися
                                    this.wakeUpEvent[0].Reset();

                                    continue;
                                }

                                //...якщо одержали сигнал до виходу з обробки...
                                if (eventIdx == 1)
                                {
                                    //...зупиняємо контрольований алгоритм
                                    this.eFileIntegrityCheck.Stop();

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

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

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

                                    return;
                                }

                                //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                                if (eventIdx == 2)
                                {
                                    //...exitимо із циклу очікування завершення
(цього й чекали в while(true)!)
                                    break;
                                }

                            } // while(true)

                        } else
                        {
                            // Скидаємо прапор коректності результату
                            this.processedOK = false;

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

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

                            return;
                        }

                        // У зв'язку із закриттям великої кількості файлових
потоків
                        // необхідно дочекатися запису змін, внесених потоком
                        // кодування в тіло класу. Потік уже не працює, але
                        // установлене їм Булевська властивість, можливо, ще
                        // "не виявилося"
                        for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                        {
                            if (!this.eFileIntegrityCheck.Finished)
                            {
                                Thread.Sleep((int)WaitTime.MinWaitTime);

                            } else
                            {
                                break;
```

```
                        }
                    }

                    // Указуємо, що основний том коректний
                    if (this.eFileIntegrityCheck.ProcessedOK)
                    {
                        dataVolIsOK = true;
                    }

                } else
                {
                    // Указуємо, що основний том коректний
                    dataVolIsOK = true;
                }

                // Виводимо прогрес обробки
                if (
                        ((dataNum % progressMod1) == 0)
                     &&
                        (OnUpdateFileAnalyzeProgress != null)
                    )
                {
                    OnUpdateFileAnalyzeProgress(((double)(dataNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                }

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

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Указуємо на те, що обробка була перервана
                    this.processedOK = false;

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

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

                    return;
                }
            }

            // Якщо даний основний том не ушкоджений, записуємо його в
"volList",
            // а інакше збільшуємо лічильник ушкоджених томів і ставимо на
місце
            // номера тому значення "-1", що вкаже на необхідність
підстановки
            // тому для відновлення
            if (dataVolIsOK)
            {
                this.volList[volListIdx++] = dataNum;

            } else
            {
                this.volList[volListIdx++] = -1;

                // Збільшуємо лічильник кількості ушкоджених основних томів
                dataVolMissCount++;
            }
        }

        // Тепер, коли знаємо кількість ушкоджених основних томів,
        // потрібно просканувати всі файли для відновлення, і визначити
```

```
            // необхідну їхню частину в список томів, а "надлишок" помістити в
            // список альтренативних томів для відновлення
            for (int eccNum = this.dataCount; eccNum < (this.dataCount +
this.eccCount); eccNum++)
            {
                // Спочатку припускаємо, що поточний том ушкоджено
                bool eccVolIsOK = false;

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

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

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

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

                                    //...програма переходить у режим сна
                                    ManualResetEvent.WaitAll(this.executeEvent);

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

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

                                //...якщо одержали сигнал до того, щоб
прокинутися -
                                // переходимо на нову ітерацію, тому що
прокидаємося
                                // перед постановкою на паузу...
                                if (eventIdx == 0)
                                {
                                    //...попередньо скинувши подію, що змусила
нас прокинутися
                                    this.wakeUpEvent[0].Reset();
```

```
                                continue;
                            }

                            //...якщо одержали сигнал до виходу з обробки...
                            if (eventIdx == 1)
                            {
                                //...зупиняємо контрольований алгоритм
                                this.eFileIntegrityCheck.Stop();

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

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

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

                                return;
                            }

                            //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                            if (eventIdx == 2)
                            {
                                //...exitимо із циклу очікування завершення
(цього й чекали в while(true)!)
                                break;
                            }

                        } // while(true)

                    } else
                    {
                        // Скидаємо прапор коректності результату
                        this.processedOK = false;

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

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

                        return;
                    }

                    // У зв'язку із закриттям великої кількості файлових
потоків
                    // необхідно дочекатися запису змін, внесених потоком
                    // кодування в тіло класу. Потік уже не працює, але
                    // установлене їм Булевська властивість, можливо, ще
                    // "не виявилося"
                    for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                    {
                        if (!this.eFileIntegrityCheck.Finished)
                        {
                            Thread.Sleep((int)WaitTime.MinWaitTime);

                        } else
                        {
                            break;
                        }
                    }

                    // Указуємо, що том для відновлення коректний
                    if (this.eFileIntegrityCheck.ProcessedOK)
```

```
                        {
                            eccVolIsOK = true;
                        }

                    } else
                    {
                        // Указуємо, що том для відновлення коректний
                        eccVolIsOK = true;
                    }

                    // Виводимо прогрес обробки
                    if (
                            ((eccNum % progressMod1) == 0)
                         &&
                            (OnUpdateFileAnalyzeProgress != null)
                        )
                    {
                        OnUpdateFileAnalyzeProgress(((double)(eccNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                    }

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

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        // Указуємо на те, що обробка була перервана
                        this.processedOK = false;

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

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

                        return;
                    }
                }

                // Якщо том для відновлення гарний...
                if (eccVolIsOK)
                {
                    //...- додаємо його в список
                    altEccList[altEccListIdx++] = eccNum;

                    // Збільшуємо лічильник кількості томів для відновлення
                    eccVolPresentCount++;

                } else
                {
                    //...а інакше вказуємо, що том ушкоджено
                    altEccList[altEccListIdx++] = -1;
                }
            }

            // Якщо значення лічильника кількості коректних томів для
відновлення збігається
            // зі значенням лічильника томів для відновлення конфігурації - всі
томи для
            // відновлення є неушкодженими
            if (eccVolPresentCount == this.eccCount)
            {
                this.allEccVolsOK = true;
            }

            // Виводимо статистику ушкоджень
```

```
            if (OnGetDamageStat != null)
            {
                // Обчислюємо загальний відсоток ушкоджень (суму ушкоджень
                // основних томів і томів для відновлення ділимо на загальну
кількість томів)
                double percOfDamage = ((double)(dataVolMissCount +
(this.eccCount - eccVolPresentCount)) / (double)(this.dataCount +
this.eccCount)) * 100;

                // Обчислюємо відсоток "" альтернативних томів, щовижили, для
відновлення
                // Альтернативні томи - це спочатку ті томи, які не планується
використовувати для відновлення
                double percOfAltEcc = ((double)(eccVolPresentCount -
dataVolMissCount) / (double)this.eccCount) * 100;

                // Виводимо статистику ушкоджень
                OnGetDamageStat(percOfDamage, percOfAltEcc);
            }

            // Якщо немає ушкоджених основних томів, просто виходимо
            if (dataVolMissCount == 0)
            {
                // Повідомляємо про закінчення процесу обробки
                if (OnFileAnalyzeFinish != null)
                {
                    OnFileAnalyzeFinish();
                }

                // Указуємо на те, що дані не ушкоджені
                this.processedOK = true;

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

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

                return;
            }

            // Якщо ми не зможемо відновити ушкодження...
            if (eccVolPresentCount < dataVolMissCount)
            {
                //...указуємо на те, що дані не можуть бути відновлені
                this.processedOK = false;

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

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

                return;
            }

            // Переміщаємося на початок списку альтернативних томів для
відновлення
            altEccListIdx = 0;

            // Тепер пробігаємося по вектору "volList", і замість кожного зі
значень "-1"
            // підставляємо чергове значення зі знайденого діапазону
            for (int i = 0; i < this.dataCount; i++)
            {
                if (this.volList[i] == -1)
                {
                    // Пробігаємося по векторі томів для відновлення,
                    // зупиняючись на коректному томі для відновлення
```

```
                    while (altEccList[altEccListIdx] == -1)
                    {
                        altEccListIdx++;
                    }

                    // Підставляємо на місце ушкодженого основного тому
                    // том для відновлення,...
                    this.volList[i] = altEccList[altEccListIdx];

                    //...забираючи використаний том зі списку альтернативних
                    altEccList[altEccListIdx] = -1;
                }
            }

            // Повідомляємо про закінчення процесу обробки
            if (OnFileAnalyzeFinish != null)
            {
                OnFileAnalyzeFinish();
            }

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

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

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

        #endregion Private Operations
    }
}
```

Файл MainForm.cs - головне вікно програми

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

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

        #region Windows Form Designer generated code

        /// <summary>
        /// Викликаємо метод для підтримки інтерфейсу
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.Windows.Forms.TreeNode treeNode1 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode2 = new
System.Windows.Forms.TreeNode("Documents and Settings", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode1});
            System.Windows.Forms.TreeNode treeNode3 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode4 = new
System.Windows.Forms.TreeNode("Downloads", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode3});
            System.Windows.Forms.TreeNode treeNode5 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode6 = new
System.Windows.Forms.TreeNode("Program Files", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode5});
            System.Windows.Forms.TreeNode treeNode7 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode8 = new
System.Windows.Forms.TreeNode("RapidDriver", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode7});
            System.Windows.Forms.TreeNode treeNode9 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode10 = new
System.Windows.Forms.TreeNode("Server", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode9});
            System.Windows.Forms.TreeNode treeNode11 = new
System.Windows.Forms.TreeNode("");
```

```
            System.Windows.Forms.TreeNode treeNode12 = new
System.Windows.Forms.TreeNode("WINDOWS", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode11});
            System.Windows.Forms.TreeNode treeNode13 = new
System.Windows.Forms.TreeNode("");
            System.Windows.Forms.TreeNode treeNode14 = new
System.Windows.Forms.TreeNode("WINDOWS.0", 18, 20, new
System.Windows.Forms.TreeNode[] {
            treeNode13});
            System.Windows.Forms.TreeNode treeNode15 = new
System.Windows.Forms.TreeNode("SYSTEM2 (C:)", 23, 24, new
System.Windows.Forms.TreeNode[] {
            treeNode2,
            treeNode4,
            treeNode6,
            treeNode8,
            treeNode10,
            treeNode12,
            treeNode14});
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(MainForm));
            this.menuStrip = new System.Windows.Forms.MenuStrip();
            this.fileToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.instrumentToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.adjustmentToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.encodingfilterToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.quickextractionToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.helpToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.separatorToolStripMenuItem = new
System.Windows.Forms.ToolStripSeparator();
            this.aboutToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.coderConfigGroupBox = new System.Windows.Forms.GroupBox();
            this.redundancyGroupBox = new System.Windows.Forms.GroupBox();
            this.redundancyMacTrackBar = new
EConTech.Windows.MACUI.MACTrackBar();
            this.allVolCountGroupBox = new System.Windows.Forms.GroupBox();
            this.allVolCountMacTrackBar = new
EConTech.Windows.MACUI.MACTrackBar();
            this.toolTip = new System.Windows.Forms.ToolTip(this.components);
            this.browser = new FileBrowser.Browser();
            this.repairButton = new System.Windows.Forms.Button();
            this.testButton = new System.Windows.Forms.Button();
            this.recoverButton = new System.Windows.Forms.Button();
            this.protectButton = new System.Windows.Forms.Button();
            this.exitToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.testspeedToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.menuStrip.SuspendLayout();
            this.coderConfigGroupBox.SuspendLayout();
            this.redundancyGroupBox.SuspendLayout();
            this.allVolCountGroupBox.SuspendLayout();
            this.SuspendLayout();
            //
            // menuStrip
            //
            this.menuStrip.Items.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.fileToolStripMenuItem,
            this.instrumentToolStripMenuItem,
            this.adjustmentToolStripMenuItem,
```

```
            this.helpToolStripMenuItem});
            this.menuStrip.LayoutStyle =
System.Windows.Forms.ToolStripLayoutStyle.Flow;
            this.menuStrip.Location = new System.Drawing.Point(0, 0);
            this.menuStrip.Name = "menuStrip";
            this.menuStrip.Size = new System.Drawing.Size(986, 21);
            this.menuStrip.TabIndex = 0;
            this.menuStrip.Text = "menuStrip";
            //
            // fileToolStripMenuItem
            //
            this.fileToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.exitToolStripMenuItem});
            this.fileToolStripMenuItem.Name = "файлToolStripMenuItem";
            this.fileToolStripMenuItem.Size = new System.Drawing.Size(45, 17);
            this.fileToolStripMenuItem.Text = "Файл";
            //
            // instrumentsToolStripMenuItem
            //
            this.instrumentToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.testspeedToolStripMenuItem});
            this.instrumentToolStripMenuItem.Name =
"instrumentsToolStripMenuItem";
            this.instrumentToolStripMenuItem.Size = new System.Drawing.Size(82,
17);
            this.instrumentToolStripMenuItem.Text = "Інструменти";
            //
            // adjustmentToolStripMenuItem
            //
            this.adjustmentToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.encodingfilterToolStripMenuItem,
            this.quickextractionToolStripMenuItem});
            this.adjustmentToolStripMenuItem.Name =
"adjustmentToolStripMenuItem";
            this.adjustmentToolStripMenuItem.Size = new System.Drawing.Size(74,
17);
            this.adjustmentToolStripMenuItem.Text = "Параметри";
            //
            // encodingfilterToolStripMenuItem
            //
            this.encodingfilterToolStripMenuItem.ImageScaling =
System.Windows.Forms.ToolStripItemImageScaling.None;
            this.encodingfilterToolStripMenuItem.Name =
"encodingfilterToolStripMenuItem";
            this.encodingfilterToolStripMenuItem.Size = new
System.Drawing.Size(216, 22);
            this.encodingfilterToolStripMenuItem.Text = "Шифруючий фільтр";
            this.encodingfilterToolStripMenuItem.Click += new
System.EventHandler(this.encodingfilterToolStripMenuItem_Click);
            //
            // helpToolStripMenuItem
            //
            this.helpToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.separatorToolStripMenuItem,
            this.aboutToolStripMenuItem});
            this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
            this.helpToolStripMenuItem.Size = new System.Drawing.Size(60, 17);
            this.helpToolStripMenuItem.Text = "Довідка";
            //
            // separatorToolStripMenuItem
            //
            this.separatorToolStripMenuItem.Name = "separatorToolStripMenuItem";
            this.separatorToolStripMenuItem.Size = new System.Drawing.Size(163,
6);
            //
```

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

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

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

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

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

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

        }

        #endregion

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

Файл ProcessForm.cs - вікно створення та перевірки raid масивів

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

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(ProcessForm));
            this.processPriorityGroupBox = new System.Windows.Forms.GroupBox();
            this.processPriorityComboBox = new System.Windows.Forms.ComboBox();
            this.processGroupBox = new System.Windows.Forms.GroupBox();
            this.processProgressBar = new System.Windows.Forms.ProgressBar();
            this.fileAnalyzeStatGroupBox = new System.Windows.Forms.GroupBox();
            this.percOfAltEccLabel = new System.Windows.Forms.Label();
            this.percOfDamageLabel = new System.Windows.Forms.Label();
            this.percOfAltEccLabel_ = new System.Windows.Forms.Label();
            this.percOfDamageLabel_ = new System.Windows.Forms.Label();
            this.logGroupBox = new System.Windows.Forms.GroupBox();
            this.logListBox = new System.Windows.Forms.ListBox();
            this.countGroupBox = new System.Windows.Forms.GroupBox();
            this.errorCountLabel = new System.Windows.Forms.Label();
            this.okCountLabel = new System.Windows.Forms.Label();
            this.errorPictureBox = new System.Windows.Forms.PictureBox();
            this.okPictureBox = new System.Windows.Forms.PictureBox();
            this.errorCountLabel_ = new System.Windows.Forms.Label();
            this.okCountLabel_ = new System.Windows.Forms.Label();
            this.toolTip = new System.Windows.Forms.ToolTip(this.components);
            this.stopButtonXP = new PinkieControls.ButtonXP();
            this.pauseButtonXP = new PinkieControls.ButtonXP();
            this.closingTimer = new System.Windows.Forms.Timer(this.components);
            this.processTimer = new System.Windows.Forms.Timer(this.components);
            this.processPriorityGroupBox.SuspendLayout();
            this.processGroupBox.SuspendLayout();
            this.fileAnalyzeStatGroupBox.SuspendLayout();
            this.logGroupBox.SuspendLayout();
            this.countGroupBox.SuspendLayout();

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

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

```
            this.SuspendLayout();
            //
            // processPriorityGroupBox
            //

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

```
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfAltEccLabel_);
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfDamageLabel_);
            this.fileAnalyzeStatGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.fileAnalyzeStatGroupBox.Location = new System.Drawing.Point(12,
216);
            this.fileAnalyzeStatGroupBox.Name = "fileAnalyzeStatGroupBox";
            this.fileAnalyzeStatGroupBox.Size = new System.Drawing.Size(459,
64);
            this.fileAnalyzeStatGroupBox.TabIndex = 0;
            this.fileAnalyzeStatGroupBox.TabStop = false;
            this.fileAnalyzeStatGroupBox.Text = "Результат аналізу цілісності
даних";
            //
            // percOfAltEccLabel
            //
            this.percOfAltEccLabel.AutoSize = true;
            this.percOfAltEccLabel.Location = new System.Drawing.Point(195, 41);
            this.percOfAltEccLabel.Name = "percOfAltEccLabel";
            this.percOfAltEccLabel.Size = new System.Drawing.Size(10, 13);
            this.percOfAltEccLabel.TabIndex = 0;
            this.percOfAltEccLabel.Text = "-";
            //
            // percOfDamageLabel
            //
            this.percOfDamageLabel.AutoSize = true;
            this.percOfDamageLabel.Location = new System.Drawing.Point(195, 20);
            this.percOfDamageLabel.Name = "percOfDamageLabel";
            this.percOfDamageLabel.Size = new System.Drawing.Size(10, 13);
            this.percOfDamageLabel.TabIndex = 0;
            this.percOfDamageLabel.Text = "-";
            //
            // percOfAltEccLabel_
            //
            this.percOfAltEccLabel_.AutoSize = true;
            this.percOfAltEccLabel_.Location = new System.Drawing.Point(7, 41);
            this.percOfAltEccLabel_.Name = "percOfAltEccLabel_";
            this.percOfAltEccLabel_.Size = new System.Drawing.Size(163, 13);
            this.percOfAltEccLabel_.TabIndex = 0;
            this.percOfAltEccLabel_.Text = "Резерв томів для відновлення:";
            //
            // percOfDamageLabel_
            //
            this.percOfDamageLabel_.AutoSize = true;
            this.percOfDamageLabel_.Location = new System.Drawing.Point(7, 20);
            this.percOfDamageLabel_.Name = "percOfDamageLabel_";
            this.percOfDamageLabel_.Size = new System.Drawing.Size(148, 13);
            this.percOfDamageLabel_.TabIndex = 0;
            this.percOfDamageLabel_.Text = "Всього пошкоджених томів:";
            //
            // logGroupBox
            //
            this.logGroupBox.Controls.Add(this.logListBox);
            this.logGroupBox.Location = new System.Drawing.Point(12, 80);
            this.logGroupBox.Name = "logGroupBox";
            this.logGroupBox.Size = new System.Drawing.Size(871, 130);
            this.logGroupBox.TabIndex = 0;
            this.logGroupBox.TabStop = false;
            this.logGroupBox.Text = "Лог процесу";
            //
            // logListBox
            //
            this.logListBox.BackColor = System.Drawing.SystemColors.Control;
            this.logListBox.BorderStyle = System.Windows.Forms.BorderStyle.None;
            this.logListBox.FormattingEnabled = true;
            this.logListBox.HorizontalScrollbar = true;
            this.logListBox.Location = new System.Drawing.Point(7, 23);
            this.logListBox.Name = "logListBox";
```

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

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

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

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

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

        }

        #endregion

        private System.Windows.Forms.GroupBox processPriorityGroupBox;
        private System.Windows.Forms.GroupBox processGroupBox;
        private System.Windows.Forms.ProgressBar processProgressBar;
        private System.Windows.Forms.GroupBox fileAnalyzeStatGroupBox;
        private System.Windows.Forms.Label percOfDamageLabel_;
        private System.Windows.Forms.Label percOfAltEccLabel_;
        private System.Windows.Forms.GroupBox logGroupBox;
        private System.Windows.Forms.GroupBox countGroupBox;
        private System.Windows.Forms.Label errorCountLabel_;
        private System.Windows.Forms.Label okCountLabel_;
        private System.Windows.Forms.ListBox logListBox;
        private System.Windows.Forms.ComboBox processPriorityComboBox;
        private System.Windows.Forms.PictureBox errorPictureBox;
        private System.Windows.Forms.PictureBox okPictureBox;
        private System.Windows.Forms.Label errorCountLabel;
        private System.Windows.Forms.Label okCountLabel;
        private System.Windows.Forms.ToolTip toolTip;
```

```
        private System.Windows.Forms.Timer closingTimer;
        private System.Windows.Forms.Label percOfAltEccLabel;
        private System.Windows.Forms.Label percOfDamageLabel;
        private PinkieControls.ButtonXP pauseButtonXP;
        private PinkieControls.ButtonXP stopButtonXP;
        private System.Windows.Forms.Timer processTimer;
    }
}
```

Файл BenchmarkForm.cs - вікно тестування швидкодії алгоритму

```
      namespace RecoveryDisk
{
    partial class BenchmarkForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.coderConfigGroupBox = new System.Windows.Forms.GroupBox();
            this.eccCountLabel = new System.Windows.Forms.Label();
            this.dataCountLabel = new System.Windows.Forms.Label();
            this.eccCountLabel_ = new System.Windows.Forms.Label();
            this.dataCountLabel_ = new System.Windows.Forms.Label();
            this.coderSpeedGroupBox = new System.Windows.Forms.GroupBox();
            this.processedDataCountLabel = new System.Windows.Forms.Label();
            this.processedDataCountLabel_ = new System.Windows.Forms.Label();
            this.timeInTestLabel = new System.Windows.Forms.Label();
            this.timeInTestLabel_ = new System.Windows.Forms.Label();
            this.benchmarkTimer = new
System.Windows.Forms.Timer(this.components);
            this.toolTip = new System.Windows.Forms.ToolTip(this.components);
            this.pauseButtonXP = new PinkieControls.ButtonXP();
            this.closeButtonXP = new PinkieControls.ButtonXP();
            this.closingTimer = new System.Windows.Forms.Timer(this.components);
            this.coderConfigGroupBox.SuspendLayout();
            this.coderSpeedGroupBox.SuspendLayout();
            this.SuspendLayout();
            //
            // coderConfigGroupBox
            //
            this.coderConfigGroupBox.Controls.Add(this.eccCountLabel);
            this.coderConfigGroupBox.Controls.Add(this.dataCountLabel);
            this.coderConfigGroupBox.Controls.Add(this.eccCountLabel_);
            this.coderConfigGroupBox.Controls.Add(this.dataCountLabel_);
            this.coderConfigGroupBox.Location = new System.Drawing.Point(12,
90);
            this.coderConfigGroupBox.Name = "coderConfigGroupBox";
            this.coderConfigGroupBox.Size = new System.Drawing.Size(212, 72);
            this.coderConfigGroupBox.TabIndex = 0;
            this.coderConfigGroupBox.TabStop = false;
```

```
            this.coderConfigGroupBox.Text = "Конфігурація кодера";
            //
            // eccCountLabel
            //
            this.eccCountLabel.AutoSize = true;
            this.eccCountLabel.Location = new System.Drawing.Point(161, 47);
            this.eccCountLabel.Name = "eccCountLabel";
            this.eccCountLabel.Size = new System.Drawing.Size(37, 13);
            this.eccCountLabel.TabIndex = 0;
            this.eccCountLabel.Text = "65535";
            //
            // dataCountLabel
            //
            this.dataCountLabel.AutoSize = true;
            this.dataCountLabel.Location = new System.Drawing.Point(161, 25);
            this.dataCountLabel.Name = "dataCountLabel";
            this.dataCountLabel.Size = new System.Drawing.Size(37, 13);
            this.dataCountLabel.TabIndex = 0;
            this.dataCountLabel.Text = "65535";
            //
            // eccCountLabel_
            //
            this.eccCountLabel_.AutoSize = true;
            this.eccCountLabel_.Location = new System.Drawing.Point(11, 47);
            this.eccCountLabel_.Name = "eccCountLabel_";
            this.eccCountLabel_.Size = new System.Drawing.Size(125, 13);
            this.eccCountLabel_.TabIndex = 0;
            this.eccCountLabel_.Text = "Томів для відновлення:";
            //
            // dataCountLabel_
            //
            this.dataCountLabel_.AutoSize = true;
            this.dataCountLabel_.Location = new System.Drawing.Point(11, 25);
            this.dataCountLabel_.Name = "dataCountLabel_";
            this.dataCountLabel_.Size = new System.Drawing.Size(89, 13);
            this.dataCountLabel_.TabIndex = 0;
            this.dataCountLabel_.Text = "Основних томів:";
            //
            // coderSpeedGroupBox
            //
            this.coderSpeedGroupBox.Controls.Add(this.processedDataCountLabel);
            this.coderSpeedGroupBox.Controls.Add(this.processedDataCountLabel_);
            this.coderSpeedGroupBox.Controls.Add(this.timeInTestLabel);
            this.coderSpeedGroupBox.Controls.Add(this.timeInTestLabel_);
            this.coderSpeedGroupBox.Location = new System.Drawing.Point(12, 9);
            this.coderSpeedGroupBox.Name = "coderSpeedGroupBox";
            this.coderSpeedGroupBox.Size = new System.Drawing.Size(212, 72);
            this.coderSpeedGroupBox.TabIndex = 0;
            this.coderSpeedGroupBox.TabStop = false;
            this.coderSpeedGroupBox.Text = "Швидкість: - Мбайт/з";
            this.coderSpeedGroupBox.Enter += new
System.EventHandler(this.coderSpeedGroupBox_Enter);
            //
            // processedDataCountLabel
            //
            this.processedDataCountLabel.AutoSize = true;
            this.processedDataCountLabel.Location = new System.Drawing.Point(55,
47);
            this.processedDataCountLabel.Name = "processedDataCountLabel";
            this.processedDataCountLabel.Size = new System.Drawing.Size(10, 13);
            this.processedDataCountLabel.TabIndex = 0;
            this.processedDataCountLabel.Text = "-";
            //
            // processedDataCountLabel_
            //
            this.processedDataCountLabel_.AutoSize = true;
            this.processedDataCountLabel_.Location = new
System.Drawing.Point(11, 47);
            this.processedDataCountLabel_.Name = "processedDataCountLabel_";
```

```
            this.processedDataCountLabel_.Size = new System.Drawing.Size(40,
13);
            this.processedDataCountLabel_.TabIndex = 0;
            this.processedDataCountLabel_.Text = "Про\'єм:";
            //
            // timeInTestLabel
            //
            this.timeInTestLabel.AutoSize = true;
            this.timeInTestLabel.Location = new System.Drawing.Point(55, 25);
            this.timeInTestLabel.Name = "timeInTestLabel";
            this.timeInTestLabel.Size = new System.Drawing.Size(10, 13);
            this.timeInTestLabel.TabIndex = 0;
            this.timeInTestLabel.Text = "-";
            //
            // timeInTestLabel_
            //
            this.timeInTestLabel_.AutoSize = true;
            this.timeInTestLabel_.Location = new System.Drawing.Point(11, 25);
            this.timeInTestLabel_.Name = "timeInTestLabel_";
            this.timeInTestLabel_.Size = new System.Drawing.Size(30, 13);
            this.timeInTestLabel_.TabIndex = 0;
            this.timeInTestLabel_.Text = "Година:";
            //
            // benchmarkTimer
            //
            this.benchmarkTimer.Interval = 1000;
            this.benchmarkTimer.Tick += new
System.EventHandler(this.BenchmarkTimer_Tick);
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // pauseButtonXP
            //
            this.pauseButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.pauseButtonXP.DefaultScheme = true;
            this.pauseButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.pauseButtonXP.Hint = "";
            this.pauseButtonXP.Location = new System.Drawing.Point(12, 175);
            this.pauseButtonXP.Name = "pauseButtonXP";
            this.pauseButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.pauseButtonXP.Size = new System.Drawing.Size(101, 23);
            this.pauseButtonXP.TabIndex = 0;
            this.pauseButtonXP.Text = "Пауза";
            this.toolTip.SetToolTip(this.pauseButtonXP, "Постановка/зняття
процесу тестування продуктивності з паузи");
            this.pauseButtonXP.Click += new
System.EventHandler(this.pauseButtonXP_Click);
            //
            // closeButtonXP
            //
            this.closeButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.closeButtonXP.DefaultScheme = true;
            this.closeButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.closeButtonXP.Hint = "";
            this.closeButtonXP.Location = new System.Drawing.Point(123, 175);
            this.closeButtonXP.Name = "closeButtonXP";
            this.closeButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.closeButtonXP.Size = new System.Drawing.Size(101, 23);
```

```
            this.closeButtonXP.TabIndex = 1;
            this.closeButtonXP.Text = "Закрити";
            this.toolTip.SetToolTip(this.closeButtonXP, "Припинення тестування
продуктивності із закриттям даного вікна");
            this.closeButtonXP.Click += new
System.EventHandler(this.closeButtonXP_Click);
            //
            // closingTimer
            //
            this.closingTimer.Tick += new
System.EventHandler(this.closingTimer_Tick);
            //
            // BenchmarkForm
            //
            this.AutoScaleDimensions = new System.Drawing.Size(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(236, 210);
            this.ControlBox = false;
            this.Controls.Add(this.pauseButtonXP);
            this.Controls.Add(this.closeButtonXP);
            this.Controls.Add(this.coderSpeedGroupBox);
            this.Controls.Add(this.coderConfigGroupBox);
            this.DoubleBuffered = true;
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "BenchmarkForm";
            this.ShowIcon = false;
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "Підготовка";
            this.Load += new System.EventHandler(this.BenchmarkForm_Load);
            this.coderConfigGroupBox.ResumeLayout(false);
            this.coderConfigGroupBox.PerformLayout();
            this.coderSpeedGroupBox.ResumeLayout(false);
            this.coderSpeedGroupBox.PerformLayout();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox coderConfigGroupBox;
        private System.Windows.Forms.Label dataCountLabel_;
        private System.Windows.Forms.Label eccCountLabel_;
        private System.Windows.Forms.Label eccCountLabel;
        private System.Windows.Forms.Label dataCountLabel;
        private System.Windows.Forms.GroupBox coderSpeedGroupBox;
        private System.Windows.Forms.Label timeInTestLabel_;
        private System.Windows.Forms.Label timeInTestLabel;
        private System.Windows.Forms.Label processedDataCountLabel;
        private System.Windows.Forms.Label processedDataCountLabel_;
        private System.Windows.Forms.Timer benchmarkTimer;
        private PinkieControls.ButtonXP closeButtonXP;
        private PinkieControls.ButtonXP pauseButtonXP;
        private System.Windows.Forms.ToolTip toolTip;
        private System.Windows.Forms.Timer closingTimer;
    }
}
```

Файл About.cs - вікно довідки про програму

```
namespace RecoveryStar
{
    partial class AboutForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.developersListBoxdevelopersListBox = new
System.Windows.Forms.ListBox();
            this.RSIconTimer = new System.Windows.Forms.Timer(this.components);
            this.okButtonXP = new PinkieControls.ButtonXP();
            this.SuspendLayout();
            //
            // developersListBoxdevelopersListBox
            //
            this.developersListBoxdevelopersListBox.BackColor =
System.Drawing.SystemColors.Control;
            this.developersListBoxdevelopersListBox.BorderStyle =
System.Windows.Forms.BorderStyle.None;
            this.developersListBoxdevelopersListBox.FormattingEnabled = true;
            this.developersListBoxdevelopersListBox.Items.AddRange(new object[]
{
            "МАГІСТЕРСЬКА ДИПЛОМНА РОБОТА",
            "",
            "На тему:",
            "",
            "Дослідження та програмна реалізація системи безперервної роботи
додатків при серверній віртуалізації",
            "",
            "",
            "",
            "Керівник: Якименко Н.М.",
            "",
            "Розробив: студент Віднічук Ганна Миколаївна",
            "                  гр. КІ 23 МЗ
",
            "",
            "м. Кропивницький 2024"});
            this.developersListBoxdevelopersListBox.Location = new
System.Drawing.Point(11, 6);
```

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

        }

        #endregion

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

    }
}
```