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

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2021 р.

**ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА**
**за другим (магістерським) рівнем вищої освіти**
на тему
**“Дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах”**

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-20М-1,4
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Шаповалов А.О.
«____» _____ 2021 р.

Керівник проекту
кандидат технічних наук
_____ Олександр ДОРЕНСЬКИЙ
«____» _____ 2021 р.
Рецензент _____
_____



м. Кропивницький

**Центральноукраїнський національний технічний університет**

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань ._12 _____*"*Інформаційні технології*"*_____

Спеціальність *123 "*Комп'ютерна інженерія*"*

Освітньо-професійна (освітньо-наукова) програма _*"*Комп'ютерна інженерія*"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2021 року

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

*Шаповалову Артему Олеговичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах*

2. Керівник роботи *Доренський Олександр Павлович, канд. техн. наук*
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 42-13 від 02.08.2021 року

3. Строк подання студентом роботи до захисту *10.12.2021 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах*

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

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

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

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

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Економічний | Савеленко Г.В. | 05.10.2021 | 14.11.2021 |
| Охорона праці | Оришака О.В. | 06.10.2021 | 16.11.2021 |
| | | | |

7. Дата видачі завдання « 6 » вересня 2021 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 | 10.10.2021 р. | |
| 2. | Постановка задачі, оформлення ТЗ | 15.10.2021 р. | |
| 3. | Розробка моделі компонента | 20.10.2021 р. | |
| 4. | Розробка структур даних | 25.10.2021 р. | |
| 5. | Розробка алгоритмів зв’язку та відображення | 30.10.2021 р. | |
| 6. | Програмування алгоритмів | 10.11.2021 р. | |
| 7. | Розрахунок економічної ефективності | 13.11.2021 р. | |
| 8. | Розрахунки з охорони праці та техніки безпеки | 15.11.2021 р. | |
| 9. | Оформлення ПЗ | 17.11.2021 р. | |
| 10. | Попередній захист роботи | 10.12.2021 р. | |
| | | | |

Дата видачі завдання
« 6 » вересня 2021 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2021 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

# АНОТАЦІЯ

**Шаповалов А.О. Дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.**

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи захисту та відновлення даних в хмарних сервісах.

Метою розробки є дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах.

Об'єктом дослідження є процес захисту та відновлення даних в хмарних сервісах.

Предметом дослідження є методи захисту та відновлення даних в хмарних сервісах.

Методи дослідження базуються на методах хмарних технологій, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи захисту та відновлення даних в хмарних сервісах.

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

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

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows XP/Vista/7/8/10.

Програму розроблено в середовищі Visual C#.

**Ключові слова**: комп'ютерна інженерія, захист даних, відновлення даних, хмарні сервіси

# ABSTRACT

**Shapovalov A.O. Research and software implementation of data protection and recovery system in cloud services. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2021**

In this final qualification work on the second (master's) level of higher education the software which is intended for system of protection and data recovery in cloud services is developed.

The purpose of development is research and software implementation of data protection and recovery system in cloud services.

The object of research is the process of data protection and recovery in cloud services.

The subject of research is methods of data protection and recovery in cloud services.

Research methods are based on methods of cloud technologies, methods of mathematical statistics, methods of software development.

The result is a software implementation of data protection and recovery system in cloud services.

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.

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

The program can be used on an IBM PC with Windows XP / Vista / 7/8/10.

The program is developed in Visual C # environment.

**Keywords:** computer engineering, data protection, data recovery, cloud services

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

| | | |
|---|---|---|
| ВМ | – | Віртуальна машина |
| НП | – | Надзвичайна подія |
| ОС | – | Операційна система |
| ПЗ | – | Програмне забезпечення |
| СЗД | – | Системи зберігання даних |
| ЦОД | – | Центр обробки даних |
| AWS | – | Amazon Web Services |
| BIOS | – | Basic Input-Output System |
| CDP | – | Continuous Data Protection – технології безперервного захисту даних |
| LINQ | – | Language Integrated Query |
| RAID | – | Redundant Array of Inexpensive Disks |
| RPO | – | Recovery Point Objective – цільова точка відновлення |
| RTO | – | Time Objective – цільовий час відновлення |

# ВСТУП

**Актуальність теми**. Усе було б дуже просто, якби дані перебували під постійним захистом від всіх видів погроз, забезпечення їхнього захисту не робило впливу на бізнес-операції й дані вдавалося миттєво відновити. Але, як і у всіх інших аспектах, що мають відношення до інформаційних технологій, при проектуванні інфраструктури ефективного захисту даних доводиться йти на певні компроміси.

Головними атрибутами оцінки ефективності рішень для захисту й відновлення даних є вікно резервного копіювання, цільова точка відновлення (Recovery Point Objective, RPO) і цільовий час відновлення (Recovery Time Objective, RTO). Ці параметри характеризують час, виділюваний для операцій резервного копіювання, інтервал між двома такими операціями й тривалість процедури відновлення, виконуваної у випадку помилки, збоїти або аварії.

Кожний з них є також мірилом простою:

– часу, протягом якого дані недоступні в процесі резервного копіювання;

– часу, що потрібно для відтворення нових, ще не зарезервованих даних;

– часу, яке потрібно затратити на операцію відновлення.

Залежно від того, яке місце в організації займають система й дані, які захищаються, тривалість простою, викликаного операціями захисту або відновлення, тим або іншим способом відіб'ється й на фінансових показниках.

Середня вартість простою в різних галузях становить 5600 доларів у хвилину, або більше 300 тис. доларів у годину. Неприступність критично важливих даних обходиться великим організаціям у мільйони доларів у годину.

Усе було б дуже просто, якби дані (особливо мають важливе значення для бізнесу) перебували під постійним захистом від всіх видів погроз, забезпечення їхнього захисту не робило впливу на бізнес-операції й дані вдавалося миттєво відновити у всіх ситуаціях. Але є й четвертий показник ефективності – вартість.

Як і у всіх інших аспектах, що мають відношення до інформаційних технологій, та й у повсякденному житті, при проектуванні інфраструктури ефективного захисту даних доводиться йти на певні компроміси, щоб:

– кожному конкретному набору даних, залежно від його цінності для організації, надавалися мінімальні рівні сервісу (вікно резервного копіювання, RPO, RTO), абсолютно необхідні для ведення бізнесу;

– організація несла мінімально можливі витрати.

Інвестиції в рішення для захисту даних найкраще розглядати як витрати на страхування, які не приносять організації доходів і інших вигід, поки не трапляється якась серйозна неприємність, після чого впливають лише на показники прибутку. Однак у дійсності всі не зовсім так.

**Мета й завдання дослідження.** Метою роботи є дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах.

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

– Огляд існуючих систем захисту та відновлення даних в хмарних сервісах.

– Дослідження системи захисту та відновлення даних в хмарних сервісах.

– Програмна реалізація системи захисту та відновлення даних в хмарних сервісах.

*Об'єктом дослідження* є процес захисту та відновлення даних в хмарних сервісах.

*Предметом дослідження* є методи захисту та відновлення даних в хмарних сервісах.

*Методи дослідження* базуються на методах хмарних технологій, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод захисту та відновлення даних в хмарних сервісах.

– Розроблено вітчизняний продукт захисту та відновлення даних в хмарних сервісах, який має більш широкі можливості, на відміну від існуючих аналогів.

**Практична цінність отриманих результатів** полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі захисту та відновлення даних в хмарних сервісах.

**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 LV Науково-технічна конференція здобувачів вищої освіти «Наука – виробництву», 2021, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №12.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах, 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

Коли ми говоримо про захист даних у хмарі, то ми маємо на увазі… Простої відповіді на це питання немає, тому що він у значній мірі виникає з тих інструментів і технологій, які використовуються. З погляду IT-фахівців, достаток потенційних векторів атаки робить захист хмарних систем куди як більше складним, ніж захист традиційних систем. Але це зовсім не виходить, що потрібно планувати захист від всіх цих факторів – при первісному формулюванні моделі погроз значна їхня частина відсівається.

Логіка така:

– По-перше, потрібно розуміти, які моделі захисту даних можуть застосовуватися в хмарі.

– По-друге, визначитися які із завдань захисту інформації будуть вирішуватися самотужки, а які – передаватися на аутсорсинг.

– І по-третє, потрібний хоча б приблизний список інструментів, які будуть використовуватися при перенесенні даних у хмару.

– І на основі цих суджень визначаються завдання, які будуть вирішуватися – усередині компанії й ззовні.

**Процеси захисту даних**

Резервне копіювання й відновлення, забезпечення безперервності бізнесу, відновлення після НП – все це теми, які напевно спливуть в обговоренні, що стосується захисту даних. Є й інші терміни – комбінація двох або більше технік. Варто погодити визначення з партнерами й вендорами – якщо ви маєте на увазі під ними різні поняття, то ви можете одержати не зовсім те що хочете.

Резервне копіювання – мабуть, найпростіший в організації процес. Це захист даних дуплікацією, створенням копії даних, які перебувають у роботі. Копія зберігається в окремому сховищі, відділеному від основного логічно й

фізично (бажано відстанню). У випадку збоїти, робота сервісу/застосунку відновлюється завантаженням резервної копії з наступним її відновленням.

Відновлення в аварійних ситуаціях – це процес відновлення працездатності всієї IT-інфраструктури (яка забезпечувала функціонування застосунку/сервісу) після того як інфраструктура стала недоступна на неприпустимий проміжок часу. Відновлюються не тільки дані, але й сервери, застосунки працюючі на серверах, схема мережних взаємодій між серверами й так далі.

Визначення безперервності бізнесу містить у собі як оперативне відновлення, так і відновлення в аварійних ситуаціях. Оперативне відновлення має на увазі відновлення роботи який-небудь із систем IT-структури. Це може бути технічна відмова – вихід з ладу жорсткого диска або поломка системи кондиціювання або неполадка, пов'язана з ПЗ – збій мережного протоколу або ушкодження бази даних. Головна відмінність від відновлення в аварійних ситуаціях у тому, що зусилля по відновленню проходять у штатному режимі і як правило не мають на увазі порушень у роботі сервісу – поки йдуть роботи, навантаження переноситься на інші сервери/площадки, у той час як при НП можливо припинення роботи сервісів до усунення НП.

Аварійні ситуації здатні нанести значну втрату репутації, даним і прибуткам, але на щастя вони відносно рідкі, більшість інцидентів можна віднести до ситуацій оперативного відновлення. Хоча деякі процедури, які застосовуються в сценаріях відновлення в аварійних ситуаціях також задіються при оперативному відновленні, але як правило в цьому немає необхідності. У більшій частині ситуацій, відбудовні роботи обмежуються відновленням даних з резервної копії.

**Визначення «хмари» і захист даних**

Національний інститут стандартів і технології визначає публічний «хмарний» сервіс як «інфраструктуру, що втримується й підтримується хмарним провайдером і надається для вільного використання всім бажаючим». Приватна «хмара», відповідно до визначення того ж NIST, це «хмарна» комп'ютерна

інфраструктура, надавана хмарним провайдером для ексклюзивного використання єдиною організацією, що включає в себе численних користувачів (наприклад бізнес-підрозділу організації). Вона («хмара») перебуває у власності й управляється організації, що користується його послугами, третьою особою, або тим і іншим у різному ступені й може розташовуватися на або поза потужностями замовника.

Захист даних може виконуватися як самотужки організацією або передаватися третім особам – провайдерам сервісів резервного копіювання (backup-as-a-service), відновлення даних (recovery-as-a-service) або відновлення в аварійних ситуаціях (disaster as-a-service). Хоча не існує ні однієї компанії, яка б надавала послуги саме під такими назвами, ці моделі існують, просто, як правило, вони включені до складу більше загальних IaaS, PaaS і так далі. У кожному разі участь третьої сторони в створенні й виконанні схеми захисту має на увазі певну (і чималу) довіра до провайдеру, що повинен бути включений до складу хмарної інфраструктури організації, причому бажано із самого початку.

Багато організацій не довіряють хмарної моделі досить, щоб переходити на неї. У такому випадку можливе залучення вендорів у рамках моделі «керованої приватної хмари». Це модель взаємодії, у рамках якої сервіс-провайдер надає певні послуги в рамках своєї хмарної інфраструктури але кожному клієнтові виділяється окремий сервер, індивідуальний набір ПЗ й послуг. Наприклад, послуги захисту даних.

Коли питання про використання хмари в організації тільки обговорюється, то завдання розподілу варто почати з інвентаризації й класифікації виконуваних завдань. Які зараз виконуються й можуть виконуватися тільки на «внутрішніх» потужностях? Які можуть бути передані на аутсорсинг? Які можна винести в «хмару»? Які пріоритети в кожного класу завдань? Як організовується весь процес захисту даних зараз і як він буде організовуватися в новій структурі роботи з даними? Спробуйте категоризувати наявні завдання по цих принципах. Чітке розуміння своїх потреб – те що вам необхідно, коли ви будете проводити переговори з вендорами, інакше можна втратитися й одержати не те що потрібно.

## 1.2 Область застосування

Хмарні технології стали основою захисту даних і двигуном розвитку інновацій. Вони дозволяють компаніям більш спокійно переносити коливання економіки, кризи й навіть форс-мажорні ситуації. Привабливість і швидкий ріст попиту на хмари пов'язані з тим, що вони дозволяють швидше й дешевше впроваджувати інновації, і досягати ринкової переваги – наприклад, такого як швидка масштабованість без капітальних вкладень в інфраструктуру, що є однією з найважливіших умов успіху стартапа або просто нового проекту. Адже на ринку перемагає не той, хто придумав ідею або створив її прототип, а той, хто перший займає ринок.

Наочно про значення хмар для бізнесу говорить обсяг і динаміка росту ринку. По оцінках Deloitte, ринок хмарних послуг, інфраструктури як послуги (IaaS) дата-центрів і хмарного програмного забезпечення досягне 547 млрд. дол. США вже до кінця 2021 року.

Примітно, що для українських і західних користувачів хмари привабливі з різних точок зору, залежно від першорядних завдань бізнесу. Компанії США і ЄС розглядають хмари як інструмент досягнення конкурентної переваги – безперебійності процесів, швидкої масштабованості, оптимізації витрат і підвищення рівня використання інноваційних технологій у бізнес-процесах. В Україні ж, основний інтерес до хмарних технологій пов'язаний з інформаційною безпекою бізнесу, захистом даних і збереженням фізичного контролю над серверами. Відповідно, поки в США хмару є стимулом росту для бізнесу, в Україні хмари використовують для втримання контролю, як над бізнесом, так і над даними.

Багато керівників не знають, що хмара дозволяє бізнесу продовжувати працювати й зберігати дані “у себе”, і паралельно страхувати бізнес від втрат за допомогою резервного копіювання важливих даних або процесів у європейські дати-центри. Така страховка для даних дозволяє бізнесу відновити роботу в

лічені хвилини після рейдерської атаки, простоїв, крадіжок і будь-яких несанкціонованих дій третіх осіб.

Наприклад, порівняно недавно стався випадок із клієнтом з фінансової галузі, що через специфіку своєї діяльності й географічного розташування (дві філії в Донбасі), вчасно запустив план аварійного відновлення даних в одному з дата-центрів у Європі. Після того, як один з філій вийшов з ладу, рішення приватної хмари клієнта дозволило зробити частковий винос даних (технічно більш відомо, як Partial Site Failover) на резервний дата-центр без зупинки діяльності. Після того, як через кілька днів був загублений контроль і над другою філією, всі робітники процеси продовжували здійснюватися на базі аварійного дата-центра в Європі. Менеджмент клієнта був готовий якщо буде потреба зробити перемикання на резервну хмарну інфраструктуру, і зміг продовжити роботу із шифрованого з'єднання з резервним дата-центром, віддалено працюючи в так званому мобільному домашньому офісі mobile home office.

Таким чином, резервне копіювання даних (BaaS) дозволяє істотно знизити ризики простою систем у випадку втрати даних, кібератак або технічних неполадок. Крім цього, хмари вже стали звичним порятунком від рейдерських захватів і інших випадків втрати фізичного контролю над даними, і навіть цілим офісом. Завжди можна запустити резервну копію даних у хмарній інфраструктурі, і таким чином, забезпечити безперервність бізнес-процесу у випадку форс-мажорних ситуацій, що для сервісного бізнесу, особливо фінансового, дозволяє уникнути паніки з боку клієнтів і зміцнити репутацію надійної компанії. Безпека даних і бізнесу є першочерговим стимулом попиту на хмарні технології в Україні, у те час як глобальний попит на хмари спонукуваний потребами розвитку й впровадження інновацій.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах, 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

**Рішення Dell EMC для резервного копіювання й відновлення в хмарі для AWS**

**Переваги Dell EMC і AWS**

Одержіть максимальну віддачу від інвестицій у сервіси AWS завдяки технологіям для захисту, мобільності даних і керування ними від підрозділу Dell EMC Data Protection Solutions. Amazon Web Services (AWS) і Dell EMC допомагають замовникам по усьому світі розгортати й масштабувати захист даних у хмарі незалежно від розміщення даних і вигідно використовувати експлуатаційні й економічні переваги хмари.

Рішення Dell EMC Data Protection допомагають замовникам захистити інвестиції в хмару:

– Перенос робочих навантажень у хмару для більшої мобільності даних.

– Захист робочих навантажень у хмарі для підвищення відказостійкості даних.

– Резервне копіювання й відновлення в хмарі для експлуатаційної й економічної ефективності.

– Відновлення після збоїв у хмарі для спокою й упевненості.

**Більше швидке й економічне резервне копіювання й відновлення в хмарі.**

**Захист даних у хмарі AWS**

Рішення Dell EMC для захисту даних допомагають замовникам одержати максимальну віддачу від інвестицій у загальнодоступну хмару. Замовники

можуть легко переміщати дані між цільовими системами зберігання AWS для відновлення після збоїв, тривалого зберігання й операційного резервного копіювання. Замовники також можуть захистити хмарні застосунки для прискорення переходу до хмари.

**Захист VMware Cloud on AWS**

Рішення Dell EMC для захисту даних призначені для сучасних програмно-визначаємих ЦОД з інтеграцією із провідною в галузі технологією VMware. Рішення Dell EMC для VMware Cloud on AWS забезпечують користувачам миттєве відновлення ВМ і масштабованість для задоволення потреб зростаючого віртуального середовища.

**Відновлення після збоїв у хмарі**

Рішення Dell EMC Data Protection дозволяють замовникам використовувати сховище AWS S3 для захисту даних від природних погроз, людського фактора й зловмисників. Відновлення даних і застосунків безпосередньо в хмарі займає лічені хвилини завдяки прозорій інтеграції з локальною інфраструктурою захисту даних.

**Тривале зберігання**

Замовники можуть прискорити свою відмову від стрічкових технологій або підвищити ефективність тривалого зберігання даних за допомогою рішень Dell EMC Data Protection і AWS.

**Керування захистом даних для AWS**

Dell EMC PowerProtect Cloud Snapshot Manager – це рішення, надаване по моделі «ПЗ як послуга», що спрощує для замовників захист робочих навантажень в AWS. При цьому воно не вимагає установки або наявності інфраструктури. Замовники можуть виконувати виявлення, оркестрацію й автоматизацію захисту робочих навантажень, використовуючи політики для безперебійного резервного копіювання й відновлення після збоїв.

**Зниження вартості захисту даних у хмарі за допомогою Dell EMC**

Організаціям, які переміщають виробничі застосунки в хмару, часто буває складно виконувати умови своїх угод про рівень обслуговування за допомогою убудованих служб захисту даних, пропонованих постачальниками хмарних рішень. Dell EMC пропонує економічне уніфіковане рішення для захисту корпоративних даних, що централізує, автоматизує й прискорює процес резервного копіювання й відновлення у всієї IT-середовищу.

**Переваги рішень Dell EMC для захисту робочих навантажень VMware**

Перед IT-службами й адміністраторами віртуальних середовищ і так коштує безліч завдань, тому захист даних VMware не повинна забирати багато часу й ресурсів. Рішення Dell EMC допомагають упоратися із цим завданням. Вони надають кошти автоматизації, інтеграції з VMware і забезпечують комплексний охват найбільшої екосистеми застосунків локально й у хмарі. Вони також мають високу продуктивність, використовують кращу в галузі функцію дедуплікації, не перевантажують мережу й не вимагають більших витрат для забезпечення захисту.

**Модернізація**

Перейдіть на сучасний програмно-визначаємий центр обробки даних і захистите своє середовище VMware за допомогою рішення з комплексним охватом. Це рішення масштабується без накладних витрат, захищає комплексну віртуалізовану екосистему, забезпечує оптимізований захист даних з використанням HCI і інтегрованих пристроїв і знижує вимоги до ємності й пропускної здатності.

**Автоматизація**

Одержите високу продуктивність і низьку вартість завдяки кращій у галузі автоматизації всього стека захисту даних. Наші рішення забезпечують високу продуктивність із низькою сукупною вартістю володіння й прискорюють резервне копіювання й відновлення, а для розгортання й налаштування проксі-сервера потрібно менш 5 хвилин.

**Трансформація**

Скористайтеся ідеальним рішенням, створеним для програмно-визначаємого ЦОД і гібридної хмари. Технологія цього рішення забезпечує розширену інтеграцію із середовищами VMware, захист хмарних робочих навантажень VMware на платформі AWS, а також самообслуговування й консолідацію контролю й автоматизації із захистом даних, що відповідає цільовому рівню обслуговування.

**Комплексний, ефективний і безперебійний захист даних для VMware**

Рішення Dell EMC для захисту даних:

– створені для програмно-визначаємого ЦОД;

– глибоко інтегровані з користувальницьким інтерфейсом VMware;

– ефективно масштабуються до більшого числа віртуальних машин без втрати продуктивності;

– захищають критично важливі застосунки з інтенсивним уведенням-виводом.

Рішення Dell EMC для захисту даних спрощують захист, масштабування й перехід до хмари, а також забезпечують комплексний захист для всього середовища VMware при низькій сукупній вартості володіння.

**Програма Future-Proof Loyalty**

Програма Future-Proof Loyalty призначена для замовників. Вона гарантує захист інвестицій завдяки набору IT-можливостей і програм світового класу, що дозволяють системам зберігання, продуктам для захисту даних і гіперконвергентній інфраструктурі Dell EMC приносити переваги протягом усього терміну служби застосунків. Програма Dell EMC використовує можливості Dell Technologies, щоб запропонувати найкращу й всеосяжну програму в галузі.

**Програма Future-Proof Loyalty забезпечує переваги для замовників**

Програма Future-Proof Loyalty призначена для забезпечення захисту інвестицій завдяки набору IT-можливостей світового класу, що дозволяють

системам зберігання, продуктам для захисту даних і гіперконвергентній інфраструктурі Dell EMC приносити переваги замовникові.

**Гарантія задоволеності**

Dell EMC гарантує роботу своїх продуктів відповідно до їхніх технічних характеристик.

**Захист інвестицій**

Dell EMC пропонує варіанти оптимізації фінансових ресурсів і дозволяє компаніям точно прогнозувати майбутні витрати й коректувати їх у міру необхідності.

**Незастаріваюча технологія**

Рішення Dell EMC дозволяють безперешкодно адаптувати IT-інфраструктуру до майбутніх вимог і одержати доступ до хмарних можливостей.

**Пропозиції програми**

**Трирічна гарантія задоволеності**

Здобуваючи сервіс Dell EMC ProSupport, ви одержуєте трирічну гарантію задоволеності для систем зберігання, засобів захисту даних і гіперконвергентної інфраструктури, тоді як галузевий стандарт передбачає гарантію строком 30 днів.

**Захист інвестицій в устаткування**

Ви завжди можете здати застарілі або конкурентні системи й одержати знижку на системи зберігання, засобу захисту даних або гіперконвергентні продукти Dell EMC наступного покоління.[2]

**Концепція Clear Price**

Концепція обслуговування Clear Price від Dell EMC передбачає чітке й прозоре ціноутворення за підтримку з докладною інформацією про підтримку по передоплаті.

**Програмне забезпечення за принципом «усе включено»**

Все необхідне для зберігання й адміністрування критично важливих даних уже включено в комплект систем зберігання, що здобуваються вами, і гіперконвергентних рішень Dell EMC.

**Безпроблемна міграція даних**

Ви можете з легкістю перейти на СЗД і гіперконвергентні рішення наступного покоління Dell EMC завдяки безпроблемній модернізації й убудованим інструментам міграції даних.

**Гарантія дедуплікації при захисті даних**

Використовуйте рекомендовані кращі практики Dell EMC, щоб забезпечити гарантоване скорочення обсягу даних в 55 разів для пристроїв захисту даних.

**Гарантії скорочення обсягу збережених даних**

Dell EMC забезпечує гарантований коефіцієнт скорочення обсягу збережених даних для ваших робочих навантажень.

**Можливості роботи в хмарі**

Dell EMC пропонує засоби хмарної мобільності, послуг, захисти й керування для ресурсів зберігання, рішення для захисту даних і гіперконвергентні продукти.

**Використання хмарних ресурсів**

Dell EMC надає гнучкі моделі споживання на вимогу для IT-інфраструктури від Dell Financial Services (DFS).

**HPE: система зберігання із захистом даних**

Система захисту даних, інтегрована у флеш-накопичувачі й створена для хмари, пропонує виняткову ефективність резервного копіювання, архівування й аварійного відновлення ваших корпоративних застосунків.

**Оптимізація засобів захисту даних для гібридних IT**

Досягнення безпрецедентної гнучкості й масштабованості при захисті критично важливих застосунків завдяки сполученню хмарних і локальних IT-рішень. Система захисту даних від HPE, інтегрована у флеш-накопичувачі й створена для хмари, забезпечує ефективне використання хмарних функцій резервного копіювання й відновлення, що відрізняються гнучкістю й економічністю, у сполученні з локальною інфраструктурою для забезпечення швидкого й надійного відновлення роботи.

**Захист даних, де б вони не перебували**

HPE поєднує краще в галузі ПЗ для захисту інфраструктури й даних із професійними послугами від HPE Pointnext Services, завдяки чому для всього середовища підприємства, від центра обробки даних до хмари, забезпечується комплексна інтегрована основа аварійного відновлення й зменшення ризиків.

**Спрощення**

Використовуйте рішення для створення інтелектуального автоматизованого багаторівневого рішення для захисту даних на основі політик і з урахуванням потреб підприємства. Забезпечте виконання угоди про рівень обслуговування завдяки найвищій швидкості резервного копіювання в масштабі хмари. Захиститеся від погроз за допомогою надійного рішення резервного копіювання й високодоступної системи зберігання, використовуваної для захисту.

**Переміщайте дані в потрібні місця**

Забезпечте повну підтримку хмари за допомогою інтегрованого рішення захисту даних. Створюйте для хмари резервні копії тільки унікальних даних, знижуючи витрати на хмарну СЗД в 20 разів. Зберігаєте зашифровані, самоописувані дані для простого аварійного відновлення в хмарі.

**Захист даних з урахуванням ваших вимог**

Трансформуйте своє рішення резервного копіювання даних, щоб перетворити реактивну політику страхування в ефективну проактивну послугу. Використовуйте екосистему кращих партнерів і модель оплати по факті споживання в масштабованому рішенні, що підійде як для невеликих віддалених офісів, так і для самих великих підприємств і постачальників послуг.

**Збереження резервних копій у хмарі**

Простота, економічність і безпека резервного копіювання в хмару HPE Cloud Bank Storage дозволяє замовникам HPE StoreOnce використовувати всі переваги недорогої зовнішньої об'єктної СЗД (локальної або хмарної) для тривалого зберігання резервних копій.

**Засоби відновлення, інтегровані у флеш-накопичувачі**

Об'єднання служб створення знімків, реплікації й резервного копіювання зі швидкістю роботи флеш-масиву. HPE Recovery Manager Central (RMC) зменшує витрати й складність і прискорює відновлення даних застосунків, які можуть розміщатися як у ЦОД, так і в хмарі. Це рішення задово льняє високим вимогам застосунків, виконуваних на флеш-масивах, і SLA із самими строгими показниками RTO/RPO.

**Захист на основі застосунків**

Підвищте продуктивність і ефективність дедуплікації для корпоративних застосунків, наприклад SAP HANA, Oracle, SQL, а також систем резервного копіювання незалежних розроблювачів, таких як Commvault, Data Protector,Veritas і Veeam. Скоротіть витрати й рівень ризику за рахунок уніфікованого керування.

**Пристрій для резервного копіювання й захисту даних HPE StoreOnce**

Інтелектуальна система зберігання, що трансформує захист даних у гібридних IT-середовищах і забезпечує простоту, підвищену продуктивність і гнучкість і при цьому коштує дешевше традиційних рішень.

**Спрощення захисту даних з гібридних IT-середовищах**

Готова для використання в хмарних рішеннях платформа захисту даних дозволяє спростити керування, скоротити витрати й зменшити ризики. HPE StoreOnce забезпечує миттєвий захист центра обробки даних, економічне архівування й відновлення після збоїв у хмарі, а широкі можливості масштабування роблять це рішення підходящої як для малих віддалених офісів, так і для великих підприємств і постачальників послуг.

**Останні інновації**

Прискорене резервне копіювання й відновлення даних завдяки простому, надійному й доступному рішенню HPE для зберігання даних, готовому до використання в хмарі.

**Спрощення операцій**

Спростите резервне копіювання в хмарі за допомогою рішення для захисту даних, що спеціально створено для хмари й не вимагає окремого шлюзу або віртуального пристрою. Ви зможете забезпечити повну підтримку хмари в рішеннях резервного копіювання й корпоративних застосунків, забезпечити інтеграцію обраного хмарного рішення, виконувати резервне копіювання в хмару тільки унікальних даних, що дозволить в 20 разів скоротити витрати на зберігання даних у хмарі1, а також зберігати дані резервних копій у зашифрованому виді зі зрозумілим описом, що спрощує відновлення хмари у випадку збоїти

**Виконання угод про рівень обслуговування**

Доможіться кращих у галузі показників по зборі даних. Ви зможете в 23 рази прискорити процеси резервного копіювання й в 15 разів – відновлення для корпоративних застосунків з мінімальним впливом на роботу виробничого середовища за допомогою прямого резервного копіювання із систем HPE 3PAR або HPE Nimble в HPE StoreOn ce з використанням HPE RMC. Забезпечте контрольований ріст даних і гарантоване скорочення витрат на резервне копіювання й загальну кількість устаткування на 95% завдяки інтелектуальної дедуплікації

**Захист даних з урахуванням ваших потреб**

Насолоджуйтеся простій і надійний захист даних з оплатою резервного копіювання як послуги з факту використання[9] і ємністю, що росте разом з вашим бізнесом. Дане рішення можна розвертати в хмарах, віртуальних і фізичних середовищах. Це гарантує захист інвестицій, виключає прив'язку до певного постачальника, а також забезпечує повну інтеграцію у всієї широкої екосистемі незалежних постачальників послуг.

**HPE StoreOnce – платформа нового покоління**

Забезпечте найвищу продуктивність у масштабі хмари, збільшення ємності до 10 разів і швидкості резервного копіювання до 3 разів. Спростіть

керування завдяки можливості контролю декількох систем за допомогою однієї панелі. Підвищте оперативність роботи завдяки гнучкій програмно обумовленому захисту.

**HPE RMC 6.2**

Функції резервного копіювання й керування даними копії, інтегровані в системи HPE 3PAR, HPE Primera і HPE Nimble Storage, забезпечують захист більшого обсягу даних, відрізняються більше високою продуктивністю й меншою вартістю. Необхідні політики для виконання угоди про рівень обслуговування можна створити за допомогою декількох клацань миші.

**Інтеграція Commvault**

Спростите резервне копіювання хмари завдяки підтримці Commvault для системи зберігання HPE Cloud Bank. Інтеграція Commvault з функціями дедуплікації на стороні джерела HPE StoreOnce і реплікацією з невеликим завантаженням каналу забезпечують більше швидке й ефективне резервне копіювання й відновлення у випадку збоїв.

**HPE StoreOnce VSA**

Потрібно ефективне резервне сховище з дедуплікацією і без обмежень, характерних для спеціалізованих пристроїв? Якщо так, HPE StoreOnce VSA допоможе задовольнити ваші потреби, маючи гнучкість віртуального пристрою. Програмно обумовлене резервне сховище підтримує всі функції спеціалізованих систем StoreOnce і гарантує сумісність програмного забезпечення для захисту даних. Крім іншого, функції переносу на іншу платформу й доступності в vSphere і Hyper-V підвищують гнучкість і відказостійкість систем. Можна настроїти до 500 Тбайт корисної ємності із кроком 1 Тбайт. Сервер ліцензій спрощує керування ліцензіями у великих або динамічних розгортаннях StoreOnce VSA. Інтелектуальне об'єднане керування через одну консоль ще більше полегшує експлуатацію декількох екземплярів StoreOnce VSA. Реплікація StoreOnce і хмарне сховище Cloud Bank Storage для додаткового захисту дозволяють переносити дані резервних копій за межі підприємства й у малозатратні служби об'єктних систем зберігання даних для тривалого економічного зберігання.

**Нові можливості**

– У системі StoreOnce VSA 4.2 розширена підтримка розгортання в середовищах VMware з високою доступністю.

– VSA 4.2 забезпечує автоматичне аварійне перемикання VSA з непрацездатного хоста на хост у режимі очікування при використанні підключень до даних по протоколах IP і FC. Завдяки цьому час простою через відмову хоста скорочується до лічених хвилин.

– Новий віртуальний пристрій StoreOnce High Availability Manager полегшує керування системами StoreOnce VSA у конфігураціях з високою доступністю.

**Можливості**

Програмно обумовлена резервна система зберігання даних ємністю до 500 Тбайт із HPE StoreOnce VSA.

У порівнянні з попереднім поколінням StoreOnce VSA показники покращилися в 10 разів, що припускає не тільки наявність додаткової ємності, але й ріст продуктивності, числа потоків, джерел реплікації й ресурсів зберігання.

Рішення StoreOnce VSA можна настроїти всього з 4 Тбайт ємності й масштабувати до 500 Тбайт із кроком 1 Тбайт, не переплачуючи за ємність, що знадобиться пізніше зі збільшенням обсягів резервного копіювання.

Доступні й інші віртуальні пристрої для дедуплікації резервної системи зберігання даних, але жодне з них не пропонує 500 Тбайт ємності. Розмір StoreOnce VSA може в 5 разів перевищувати розмір наступного найбільшого віртуального пристрою для дедуплікації.

**Гнучке налаштування продуктивності і ємності**

Гнучке додавання нової ємності із кроком 1 Тбайт або будь-яким значенням, кратним 1 Тбайт. Ліцензії додаються безпосередньо до екземпляра VSA або із сервера ліцензій HPE StoreOnce VSA.

Програмно обумовлена продуктивність із ресурсами, виділеними для масштабування пропускної здатності, ємності, потоків і числа цільових об'єктів

резервного копіювання, які необхідні для дотримання графіків резервного копіювання й угод про рівень обслуговування.

Можна використовувати кілька екземплярів VSA, щоб задати різну продуктивність резервного копіювання. Наприклад, можна мати VSA з великою ємністю, але невисокою продуктивністю, або високопродуктивний пристрій VSA з меншою ємністю. Це дозволить оптимально використовувати час, виділений для резервного копіювання.

**Вибір гіпервізору й опцій оцінки**

Запустите HPE StoreOnce VSA в VMware vSphere і Microsoft Hyper-V* або в обох гіпервізорах за допомогою єдиної консолі, щоб управляти екземплярами VSA. Примітка. Hyper-V у даний момент не підтримується кодом VSA версії 4.2

Пробна версія HPE StoreOnce VSA припускає до 90 днів використання. Вона допоможе оцінити всі можливості цього рішення. Щоб продовжити використання з усіма збереженими конфігураціями, придбайте ліцензію. Див. сторінку https://www.hpe.com/storage/TryStoreOnceVSA.

Безкоштовна версія ПЗ StoreOnce VSA на 1 Тбайт для підтримки розширеної оцінки й використання некритичних продуктів з можливістю самопідтримки, яку можна легко обновити для додавання підтримки HPE, хмарного сховища Cloud Bank Storage, шифрування й до 500 Тбайт ємності. Див. сторінку www.hpe.com/storage/freebackup.

**Централізоване керування ліцензіями HPE StoreOnce VSA для великих і динамічних розгортань**

У динамічних або великомасштабних системах StoreOnce VSA керування ліцензіями може займати багато часу, а сервер ліцензій StoreOnce VSA заощаджує час завдяки централізації керування ліцензіями. Ліцензії додаються на сервер ліцензій AutoPass для формування пула ресурсів, виділюваного підключеним VSA у міру необхідності.

За допомогою сервера ліцензій AutoPass можна повертати ліцензії зі списаних екземплярів StoreOnce VSA на сервер ліцензій і використовувати

повторно на нових пристроях StoreOnce VSA, щоб скоротити пов'язані з ліцензуванням витрати.

У сполученні з об'єднаним керуванням StoreOnce сервер ліцензій StoreOnce VSA забезпечує керування великими розгортаннями StoreOnce VSA у єдиній консолі.

## 2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

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

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

що може легко використовуватися в клієнтському коді. В Visual C# 5.0 вираження LINQ (Language-Integrated Query) роблять строго-типізований запит першокласною конструкцією мови.

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

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

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

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

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

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

Якщо буде потрібно забезпечити взаємодію з іншим програмним забезпеченням Windows, таким як об'єкти COM або власні бібліотеки DLL Win32, у мові Visual C# можна використовувати процес, що називається "Interop". Процес Interop дозволяє програмам на Visual C# виконувати практично будь-які дії, які може виконувати вихідний додаток на C++. Мова Visual C# підтримує

навіть покажчики й поняття "небезпечного" коду для тих випадків, коли прямий доступ до пам'яті має вкрай важливе значення.

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

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

### 2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи захисту та відновлення даних в хмарних сервісах.

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

Стратегія захисту й резервного копіювання даних повинна враховувати розширення сфери IT, у яку тепер входять не тільки ЦОДи, але й публічні хмари, а також периферійні обчислення.

Резервне копіювання й захист даних зараз важливіше, ніж коли-або, оскільки ми вступили в епоху хмар і рухаємося до усе більше розосередженого екосистемі IT. Але в будь-який екосистемі надійний план резервного копіювання повинен базуватися на аудиті підметів захисту даних і на процесах, які можуть бути використані для забезпечення їхньої безпеки.

**Масштаби захисту даних**

Захист даних охоплює широкий спектр сценаріїв, у т.ч. наступні:

– ушкодження – системне ПЗ або застосунки випадково змінюють контент;

– помилка користувача – користувачі ненавмисно видаляють дані;

– відмова устаткування – різного роду проблеми з носіями інформації, збої серверів і т.п.;

– втрата устаткування – його вихід з ладу в результаті пожежі, повені або крадіжки;

– зловмисне знищення – різні дії, спрямовані на видалення даних або відмову в доступі до них, наприклад, за допомогою вимагацького ПЗ.

Хоча більшість таких сценаріїв ставляться до приватних ЦОДів, що розширюється використання публічних хмар означає, що ці середовища також повинні бути захищені.

Отже, для захисту даних варто використовувати застосунки, що запускаються на власній площадці, а також працюючі в якості платформи хмарного ПЗ, такі як Office365 або SalesForce.com.

Сервіси публічних хмар начебто названих вище не роблять резервне копіювання даних за замовчуванням, тільки для відновлення після системних збоїв. Тому відновлення повідомлень електронної пошти після їхнього видалення лягає на власника даних і повинне бути передбачене планом захисту.

Оскільки величезна інфраструктура доступна через Інтернет, IT-підрозділам варто також подбати про захист від витоків даних (DLP).

**Дотримання рівнів обслуговування**

Задовольнити потреби бізнесу в забезпеченні безпеки даних можна завдяки застосуванню різних рівнів обслуговування до захисту й відновлення даних. Іншими словами, параметри відновлення визначають мети захисту.

Двома головними критеріями є час відновлення (Recovery Time Objective, RTO) і точка відновлення (Recovery Point Objective, RPO). Останній визначає обсяг припустимої втрати даних. Наприклад, RPO=0 означає, що повинні бути відновлені всі дані, які були наявні на момент збою.

**Реплікація або резервне копіювання**

Захист даних виробляється за допомогою ряду доступних технологій.

Реплікація – це процес створення безлічі надлишкових копій даних розраховуючи на те, що хоча б одна з них збережеться після катастрофи.

Звичайно реплікація здійснюється за допомогою синхронного або асинхронного копіювання з використанням масиву пам'яті. Але копіювання може вироблятися на рівні гіпервізору й застосунку. Платформи баз даних уже надають можливість реплікації за допомогою таких інструментів, як доставка журналів (log shipping). Платформи NoSQL дозволяють робити реплікацію по моделі погодженості в остаточному підсумку (eventual consistency).

Варто пам'ятати, що тільки реплікації недостатньо для повного захисту даних. Синхронна реплікація, наприклад, створить репліку ушкоджених даних. А асинхронна репліка не відбиває останніх змін.

**Погодженість в остаточному підсумку**

Погодженість в остаточному підсумку застосовна також до розподіленого зберігання. Наприклад, до об'єктних сховищ.

Коли затримка не має великого значення, дані можна розподілити географічно за допомогою алгоритмів з кодом надмірності (erasure coding) і асинхронного реплікування у фоновому режимі, іменованого погодженістю в остаточному підсумку.

Одним з головних переваг використання коду надмірності є можливість відновлення даних з підмножини захищеного контенту без створення додаткових повних копій.

Код надмірності здатний захистити, наприклад, від втрати даних у чотирьох точках при збільшенні ємності зберігання всього на 25%. Це дозволяє здійснювати захист багатохмарних об'єктних сховищ, що поширюється одночасно на кілька хмарних провайдерів і власні ЦОДи підприємств.

**Моментальні знімки й відстеження змінених блоків даних**

У найбільш сучасних платформах резервного копіювання застосовуються створення моментальних знімків і відстеження змінених блоків даних (changed block tracking, CBT).

Моментальні знімки фіксують дані застосунків на певний момент часу й робляться відповідно до графіка, звичайно під час пауз в операціях уведення-виводу застосунків, щоб гарантувати цілісність даних.

CBT дозволяє одержати доступ до потоку змінених даних у джерела замість того, щоб копіювати весь набір даних і займатися його дедуплікацією за допомогою пристрою або ПЗ резервного копіювання.

Найбільш очевидним є застосування моментальних знімків і CBT на віртуальних серверах, де гіпервізори типу VMware vSphere надають доступ до

потоку змінених із часу останнього моментального знімка даних. Потім ПЗ резервного копіювання створює синтезовані образи з повних і часткових резервних копій для відновлення даних у майбутньому.

Крім того, CBT добре працює з гіперконвергентними рішеннями, де потік змінених даних забезпечується на інтегрованому рівні зберігання. Nutanix Acropolis надає таку можливість для блокових сховищ, прик ріплених до віртуальних машин, і для даних, що зберігаються в Nutanix Files.

CBT і моментальні знімки дозволяють набагато швидше обробляти резервні копії даних, особливо в мережах резервного копіювання. Провайдери публічних мереж надають можливість виготовлення моментальних знімків на своїх платформах для підключених до віртуальних машин пристроїв блокового зберігання. Моментальні знімки даних переносяться на більше дешеві носії для довгострокового зберігання.

**Хмари**

Публічні хмари створюють деякі нові пробл еми й можливості для захисту даних.

Як вказувалося вище, застосунки захищають дані, використовуючи моментальні знімки.

Публічні хмари можуть використовуватися для зберігання даних, що захищаються, і архівування моментальних знімків. Вони також відмінно підходять для зберігання резервних копій завдяки можливості доступу з будь-якої географічної точки й убудованому захисту від втрати даних і збоїв. Немає необхідності піклуватися про масштабування сховища резервних копій, оскільки провайдер гарантує практично необмежену масштабованість.

Але використання публічних хмар для зберігання резервних копій має деякі недоліки.

По-перше, що досягається за допомогою дедуплікації даних економія обсягу не передається клієнтові. Тому ПЗ резервного копіювання повинне робити дедуплікацію до запису даних у хмару.

По-друге, варто враховувати проблему продуктивності. Швидкість відновлення даних з публічної хмари залежить від наявної смуги пропущення й часто обмежується провайдером.

Зараз більшість провайдерів підтримують зберігання резервних копій у публічних хмарах. У багатьох випадках програмно-апаратні рішення (які доповнюють розглянуті нижче апаратні рішення) можуть використовуватися також для відновлення даних у публічній хмарі як первинній системі зберігання. Це означає, що п ублічні хмари здатні заміняти традиційні площадки, призначені для відновлення після катастроф.

**Устаткування**

Устаткування, що дозволяє використовувати публічну хмару в якості репозиторія резервних копій, розгортається на власній площадці підприємства. Воно кешує дані локально, тоді як архівування колишніх моментальних знімків і резервних копій відбувається на недорогих системах, таких як об'єктні сховища, на власній площадці або в публічній хмарі.

Оскільки відновлення даних звичайно виробляється за допомогою найбільш свіжої версії резервної копії, архівування в публічній хмарі за допомогою апаратного рішення є економічним і дозволяє IT-підрозділам дотримувати угод про рівень обслуговування.

**Вимагацьке ПЗ й безпека**

Новим є розгортання застосунків у публічних ЦОДах і на периферії обчислювальних середовищ. Для надання локальних сервісів часто використовуються невеликі ЦОДи або машинні зали. При такому розосередженні обчислювальної потужності й даних набагато більше систем піддаються небезпеки зараження вимагацьким ПЗ.

При атаках з використанням вимагацького ПЗ хакери встановлюють на зламані сервери або ПК код, що шифрує локальні дані, і вимагають викупу за їхнє розблокування й надання до них доступу.

Виробники систем зберігання допомагають захиститися від подібних атак за допомогою простого відновлення даних з резервних копій, а також допомагають визначити, де відбулася атака, за допомогою відстеження змінених даних у відведене на резервне копіювання час.

Захист від вимагацького ПЗ – тільки частину необхідної сьогодні захисту даних. Резервні копії повинні зашифровуватися й надійно захищатися при мінімальному контрольованому доступі тільки до тих з них, які використовуються для відновлення даних.

**Формування стратегії**

Як міняється захист даних? Які стратегії варто ви користовувати IT-підрозділам?

Якщо колись захист даних будувалася навколо поділюваного зберігання й гіпервізору, то сьогодні методи захисту набагато більше різноманітні.

Тому захист вимагає безлічі процесів резервного копіювання й спостереження за статусом резервних копій, що зберігаються на власній площадці й поза нею.

Така стратегія повинна бути прив'язана до мобільності даних, оскільки в майбутньому застосунку можуть стати набагато більше мобільними й мати потребу в крос-платформному відновленні даних.

Імовірно, головною проблемою для адміністраторів зберігання й фахівців із захисту даних у найближчі роки буде домогтися, щоб дані застосунків можна було зробити доступними або вчасно їх відновити незалежно від того, де виконується робоче навантаження.

Таким чином, у майбутньому акцент, імовірно, буде зроблений на самих даних, щоб забезпечити їхню постійну доступність незалежно від платформ, на яких вони зберігаються.

### 3.2 Розробка структурної схеми

В умовах збільшення числа серверів, систем зберігання, мережних платформ і застосунків, а також територіально розподіленого розміщення ресурсів IT-інфраструктура стає дуже складною. Розвиток технологій у сполученні із щорічним ростом обсягів даних на 40-50% і ще більш твердими вимогами до рівня сервісу з боку за цікавлених осіб надзвичайно ускладнило завдання захисту даних. На структурній схемі (рисунок 3.1) показані можливі ситуації, які необхідно враховувати при складанні плану захисту й відновлення даних.

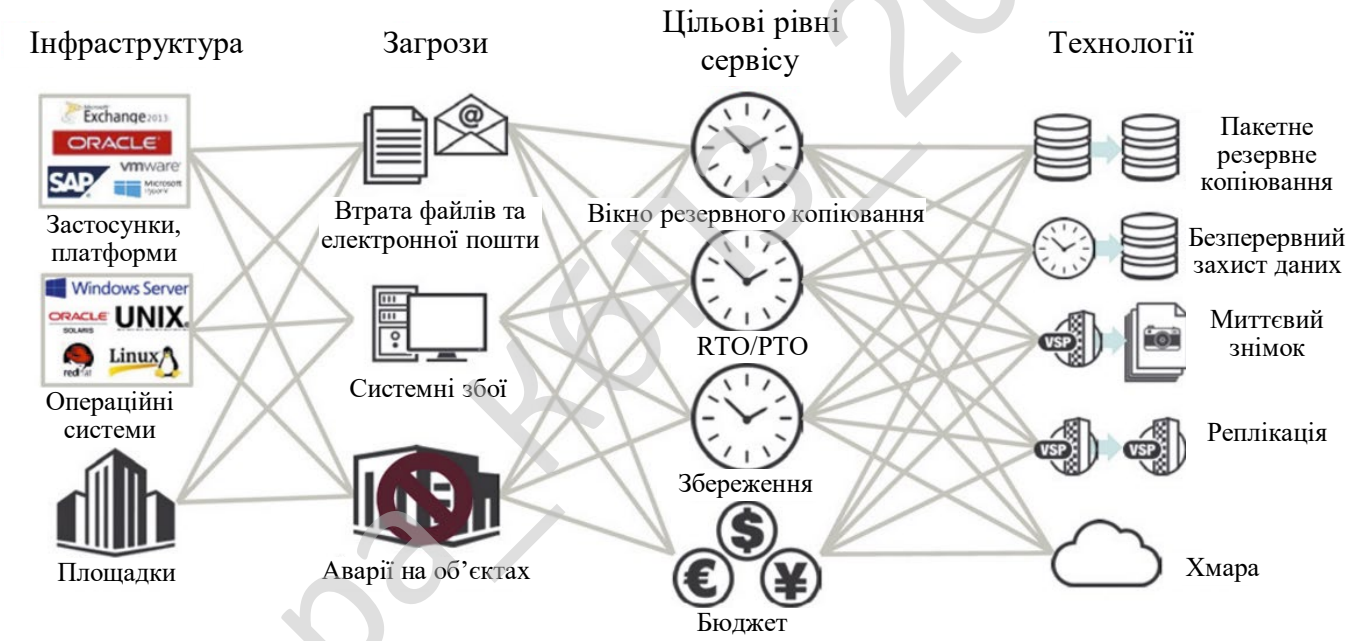


Рисунок 3.1 – Структурна схема системи

Адміністратори резервного копіювання зіштовхуються з масою інфраструктурних складностей. Кожний компонент інфраструктури необхідно захищати від різноманітних погроз і враховувати при цьому цільові рівні сервісу кожного підрозділу-бізнесу-підрозділу. Вони визначають політики, яким повинен випливати набір рішень для захисту даних, що, у свою чергу, приводить до розширення спектра використовуваних технологій.

У контексті захисту й відновлення даних часто говорять про «вікно резервного копіювання», під яким розуміють тривалість виконання операцій резервного копіювання. З технічної точки зору це невірно. Насправді вікно резервного копіювання являє собою повний інтервал (з урахуванням часу запуску й зупинки), необхідний для виконання резервного копіювання певної системи або набору даних.

Вікно резервного копіювання – один із чотирьох цільових показників, використовуваних для оцінки ефективності процесу захисту даних. До трьох іншим ставляться цільова точка відновлення, цільовий час відновлення й загальних витрат.

Якщо виконання резервного копіювання займає більше часу, ніж припустиме вікно резервного копіювання, можливі наслідки для бізнесу очевидна: або процедура резервного копіювання зупиниться до моменту її завершення з ризиком втрати важливих або навіть критичних даних, або її дозволено буде продовжити. У кожному разі це відіб'ється на рівні готовності системи, що захищається, а отже, і на багатьох операціях.

Найчастіше вікно резервного копіювання представляється неминучим злом. Бізнес визнає необхідність такого простою. Створення резервної копії має важливе значення для виживання організації, незважаючи на те що знижує операційну ефективність і рентабельність. Якщо підприємству вдасться скоротити тривалість створення резервних копій і зменшити або навіть ліквідувати вікно резервного копіювання, у нього з'явиться можливість відшкодувати упущену вигоду.

Інша категорія витрат, пов'язаних з тимчасовими витратами, обумовлена часом, що адміністратор витрачає на підтримку операцій резервного копіювання. Колись резервне копіювання виконувалося вручну і являло собою дуже трудомістку операцію. Адміністраторові резервного копіювання було потрібно:

– виконати необхідну установку й внести зміни в конфігурацію для кожної нової системи або користувача;

– маркірувати щодня записувані стрічки, після чого запустити процедуру резервного копіювання й проконтролювати її виконання;

– витягти стрічки для їхнього транспортування в центр післяаварійного відновлення або сховище;

– стерти інформацію зі стрічок, на яких зберігаються неактуальні резервні копії, для їхнього повторного використання;

– у випадку збоїти усунути неполадки й запустити знову процедуру створення резервної копії (якщо вікно резервного копіювання дозволяє це).

Інтелектуальні політики, автоматизація роботи систем, технології автоматичного виявлення, використання дискових сховищ і реплікація в центр післяаварійного відновлення дозволяють мінімізувати прикладені зусилля й скоротити витрати. Ці функції й підходи допомагають виключити більшу частину ручної праці й скоротити час, затрачуваний на виконання операцій по захисту даних. Однак багато компаній усе ще додержуються старих методів, відмовляючись від резервного копіювання або передаючи ці функції незалежним постачальникам послуг.

Щоб підвищити рівень готовності даних, знизити ризики й скоротити витрати, всі організації – як великі, так і малі – повинні застосовувати сучасні технології захисту даних.

За допомогою пропонованих рішень вікно резервного копіювання можна мінімізувати або навіть виключити. Такі результати досягаються за рахунок повністю інтегрованих технологій безперервного захисту даних (Continuous Data Protection, CDP) на рівні блоків і погоджених з додатками моментальних знімків, які робляться за допомогою спеціальних апаратних засобів. У результаті усувається необхідність сканування файлової системи для пошуку інкрементних змін і скорочується тривалість копіювання даних – до декількох секунд.

**Постійне інкрементне резервне копіювання**

Засоби створення моментальних знімків і CDP обробляють тільки нові й змінені дані. Використання інкрементної моделі на постійній основі виключає

потреба в періодичному повнім резервному копіюванні (або синтетичному повнім резервному копіюванні), властивій більшості інших рішень. Залежно від швидкості зміни даних і тривалості їхнього зберігання такий підхід дозволяє зменшити обсяг збережених резервних копій більш ніж на 90%.

Припустимо, звичайне підприємство, працює п'ять днів у тиждень і 50 тижнів у році. Їм накопичено 100 Тбайт даних, середня швидкість зміни яких становить 10% (10 Тбайт) у д ень і 50% (50 Тбайт) у тиждень. Резервні копії для оперативного відновлення зберігаються 12 тижнів, а дані, потребуючі більше тривалого зберігання, архівуються.

Це крайній випадок, що ілюструє відмінності традиційної й постійної інкрементної моделі. У типовому середовищі загальний обсяг змінених даних звичайно становить 50% у рік (50 Тбайт), що еквівалентно 1% у тиждень (1 Тбайт) і 0,2% у день (200 Гбайт).

При комбінації повного й інкрементного резервного копіювання, а також у використовуваної Hitachi Data Instance Director (HDID) моделі постійного інкрементного копіювання обсяг щоденної копії становить 10 Тбайт. Однак, у першому випадку у вихідні резервуються 100 Тбайт, тоді як у другому ніяких копій не створюється.

З урахуванням створення первісної повної резервної копії (100 Тбайт) загальний обсяг резервних копій за 12 тижнів складе:

– повне й інкрементне резервне копіювання: 1900 Тбайт (1,9 Пбайт);

– постійне інкрементне резервне копіювання: 700 Тбайт (0,7 Пбайт).

Завдяки дедуплікації даних постійне інкрементне резервне копіювання дозволяє скоротити необхідну ємність зберігання на 63% без додаткових фінансових витрат і зниження продуктивності системи. У скільки обійдуться придбання, керування й підтримка сховища з резервними копіями обсягом 1,2 Пбайт? По суті, це 2,4 Пбайт додаткові простори, тому що ми хочемо реплікувати сховище з резервними копіями в центр післяаварійного відновлення. Якщо дані з копіями зберігаються довше трьох місяців, економія виявиться ще більшою.

У типовому ж середовищі, де за рік міняєт ься 50% даних, для зберігання традиційних резервних копій протягом 12 тижнів знадобиться 1,3 Пбайт, а для зберігання постійних інкрементних копій – усього 112 Тбайт. Таким чином, економія досягне 91%. Наведене зменшення обсягів ще раз підтверджує, що при традиційному резервному копіюванні майже всі зберігати^ся данние, що, надлишкові.

Керування вікном резервного копіювання – це перший крок до зниження витрат і ризиків, пов'язаних із захистом даних, до забезпечення більше ефективного післяаварійного відновлення й поновлення операційної діяльності.

**Як RPO впливає на RTO?**

Цільова точка відновлення (RPO) указує на прийнятну для підприємства періодичність створення резервних копій і, таким чином, визначає момент часу, у який можливе відновлення даних. Якщо RPO рівняється 24 год, виходить, однієї операції резервного копіювання в день цілком достатньо. Крім того, даний показник характеризує:

– частоту виконання операцій резервного копіювання;

– обсяг нових даних, які підприємство ризикує втратити.

RPO принципово відрізняється від цільового часу відновлення (RTO). За допомогою RTO можна зрозуміти, як довго буде виконуватися процедура відновлення системи або застосунки або відновлення доступу до набору даних після незапланованої події, викликаного помилкою людини, збоєм устаткування або природним катаклізмом.

RTO визначає, яка тривалість простою (а отже, грошові втрати, ризики й упущена вигода), з яким організація готова миритися у випадку збоїти або аварії. Найчастіше для різних типів даних і видів збоїв установлюється різний цільовий час відновлення – наприклад, друга година для втраченого файлу або електронного листа, шоста година для запуску сервера, що відмовив, і два дні на відновлення операцій у випадку збоїти, що затронули весь об'єкт.

Оскільки RPO і RTO – принципово різні поняття, багатьох цікавить, чи роблять вони вплив один на одного. Як правило, на це питання відповідають негативно, але спосіб досягнення RPO самим безпосереднім образом відбивається на дотриманні RTO.

Представимо, що у вас дуже більша база даних, резервну копію якої можна створити тільки за довгі вихідні. Щоб зменшити RPO до 24 год, щоночі необхідно робити резервні копії журналів бази даних або журналів повторного виконання. У результаті можна відновити останню повну копію бази даних, а потім повторно виконати всі транзакції, збережені в журналах бази даних або журналах повторного виконання.

Число й розміри файлів, які потрібно відновити й використовувати поряд з файлами бази даних, можуть рости дуже швидко, особливо якщо ви маєте справу з масштабним кластерним середовищем на зразок Oracle Real Application Clusters (RAC). Отже, чи буде час, затрачуване на відновлення останньої повної резервної копії й всіх журналів, відповідати відведеному для великої системи баз даних RTO? Відповідь, мабуть, негативний, якщо тільки RTO не виміряється тижнями й місяцями. Така методологія захисту бази даних може бути використана для створення прийнятних цільових точок відновлення, але вона не підходить для дотримання прийнятного цільового часу відновлення.

Схожу ситуацію ми спостерігаємо й при традиційному повному + інкрементном резервному копіюванні, описаному раніше. При такій моделі повна резервна копія звичайно створюється кожні вихідні, а інкрементна – щодня протягом робочого тижня. Якщо збій відбувся в понеділок і потрібно виконати повне відновлення, ніяких труднощів це викликати не повинне: дані відновлюються з останньої резервної копії, зробленої у вихідні.

Якщо ж збій відбудеться в п'ятницю, потрібно відновити повну резервну копію, зроблену в попередні вихідні, а потім послідовно всі інкрементні набори з понеділка по четвер. У п'ятницю процедура відновлення буде виконуватися значно довше, ніж у понеділок. Чи враховується ця обставина в RTO? Крім того,

відновлення наприкінці тижня – набагато більше ризикований процес, що складається з декількох етапів, виконуваних вручну. Можливо, деякі з відновлюваних даних прийдеться переписувати до чотирьох разів.

Очевидно, що в міру подальшого збільшення обсягів даних і ускладнення IT-систем використовувані підходи прийде поліпшувати, щоб забезпечити дотримання вимог до резервного копіювання (RPO) і відновленню (RTO). Компанія Hitachi пропонує рішення, здатне захистити великі бази даних і критично важливі застосунки й значно поліпшити показники RPO і RTO. Воно містить у собі три складові:

– Моментальні знімки й технології реплікації на базі сховища, які:

– виключають із системи керування базами дані операції по захисту даних;

– усувають необхідність у вікні резервного копіювання й пов'язані з ним простої;

– дозволяють виконувати операції резервного копіювання набагато частіше, скорочуючи обсяги нових даних, піддані ризику втрати, на 90% і більше.

– Моментальні знімки й програмне забезпечення реплікації для застосунків і баз даних, які:

– переводять бази даних і застосунку в готове до резервного копіювання (відключене) стан;

– створюють у сховище моментальний знімок, після чого база даних і додаток звільняються для нормального функціонування;

– забезпечують швидке й повністю погоджене відновлення операційної діяльності протягом декількох хвилин, а не тижнів;

– Сервіси оцінки й впровадження, які визначають і конфігурують оптимальне рішення для унікального середовища підприємства.

**RPO – можлива схована вартість RTO**

Що ставиться до RTO? Залежно від конкретного визначення сюди можуть увійти деякі або навіть усе з наступних складових:

– тривалість вивчення й діагностики події;

– тривалість виконання коригувальних дій: установка нового сервера, заміна диска, відсторонення співробітника, що став причиною неполадок, переклад операцій у резервний центр;

– тривалість переустановки операційної системи й застосунків при виникненні такої необхідності;

– тривалість відновлення всіх потрібних даних з резервної копії або системи післяаварійного відновлення;

– час, витрачений на запуск і тестування відновленого середовища.

Все це виливається в дуже тривалу процедуру й приводить до простоїв. Протягом певного проміжку часу якась частина підприємства не може займатися виробничою діяльністю, що впливає на валовий дохід або прибуток або на те й інше.

Крім того, є параметр, що часто залишається за рамками зазначеного списку, але при цьому самим безпосереднім образом відбивається на тривалості повного відновлення й загальної вартості відновлення. Мова йде про цільову точку відновлення (RPO). Якщо RPO рівняється 24 год (як пр авило, резервне копіювання виконується вночі), то це означає, що ви готові примиритися із втратою нових даних, отриманих протягом доби.

Найчастіше RPO вибирається із практичних міркувань: наприклад, певну систему можна відключити тільки на ніч або на вихідні. Разом з тим RPO варто визначати з урахуванням вимог бізнесу, а не тільки виходячи з обмежень наявного програмного забезпечення для резервного копіювання.

Уявимо собі, що параметр RPO дорівнює 24 год, а збій системи відбувся о шостій годині вечора, при цьому всі дані, що втримуються там, віддалені або знищені. Можна, звичайно, відновити їх з останньої резервної копії, але вся інформація, створена й змінена після цього, буде загублена.

Ви готові упокоритися із втратою цих даних? Можливо, там присутні кілька великих замовлень із системи продажів, результати проектування за день і багато інші важливі для організації відомості. Просто знизаєте плечима й рушите

далі? Звичайно ні. Дані необхідно відновити, тобто ввести заново. Процес цей зажадає якогось часу, протягом якого співробітники могли б займатися творчою діяльністю, що знов-таки впливає на ефективність бізнесу протягом усього періоду відновлення.

Таким чином, ніж більше інтервал між операціями резервного копіювання (RPO), тим більше даних прийде відновлювати у випадку збоїти й тем вище витрати. Причому це можуть бути не просто матеріальні витрати. Представте тільки, що ви звертаєтеся до клієнта із проханням повторити раніше зроблене замовлення на мільйон доларів, тому що ваша система дала збій!

Міркуючи над цією головоломкою, нескладно прийти до бажаного виводу: необхідно скоротити час, а виходить, і гроші, які доводиться затрачати в процесі відновлення після будь-якого збою. Для цього потрібно:

– значно зменшити вікно резервного копіювання, що обмежує частоту операцій копіювання (RPO);

– істотно збільшити частоту операцій копіювання, щоб якнайменше даних піддавалося ризику втрати, після якої їх доводиться вводити заново;

– прискорити виконання операцій відновлення – як локальних (відновлення операційної діяльності), так і віддалених (післяаварійне відновлення).

### 3.3 Розробка функціональної схеми

У даній роботі для захисту та відновлення даних в хмарних сервісах використовується технологія RAID. Надамо її опис. Відразу варто помітити, що RAID розшифровується як Redundant Array of Independent Disks – надлишковий масив незалежних дисків. Споконвічно RAID розшифровувався як Redundant Array of Inexpensive Disks – надлишковий масив недорогих дисків. Під недорогими малися на увазі диски, призначені для використання в ПК, – на противагу дорогим дискам для мейнфреймів. Але так як в RAID -масивах стали

використовувати SCSI-вінчестери, які були істотно дорожче застосовуваних у більшості комп'ютерів дисків IDE, слово "недорогий" було замінено на "незалежний".

Принцип функціонування RAID-системи полягає в наступному: з набору дискових накопичувачів 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 без відказостійкості (Striped Disk Array without Fault Tolerance)**

Дисковий масив без надлишкового зберігання даних. Інформація розбивається на блоки, які одночасно записуються на окремі диски, що забезпечує підвищення продуктивності. Такий спосіб зберігання інформації ненадійний, оскільки 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 із відзеркалюванням (Mirroring & Duplexing)**

Дисковий масив 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 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 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 файлам и 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 н ьому 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 грунтується 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.

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 дис ків 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.

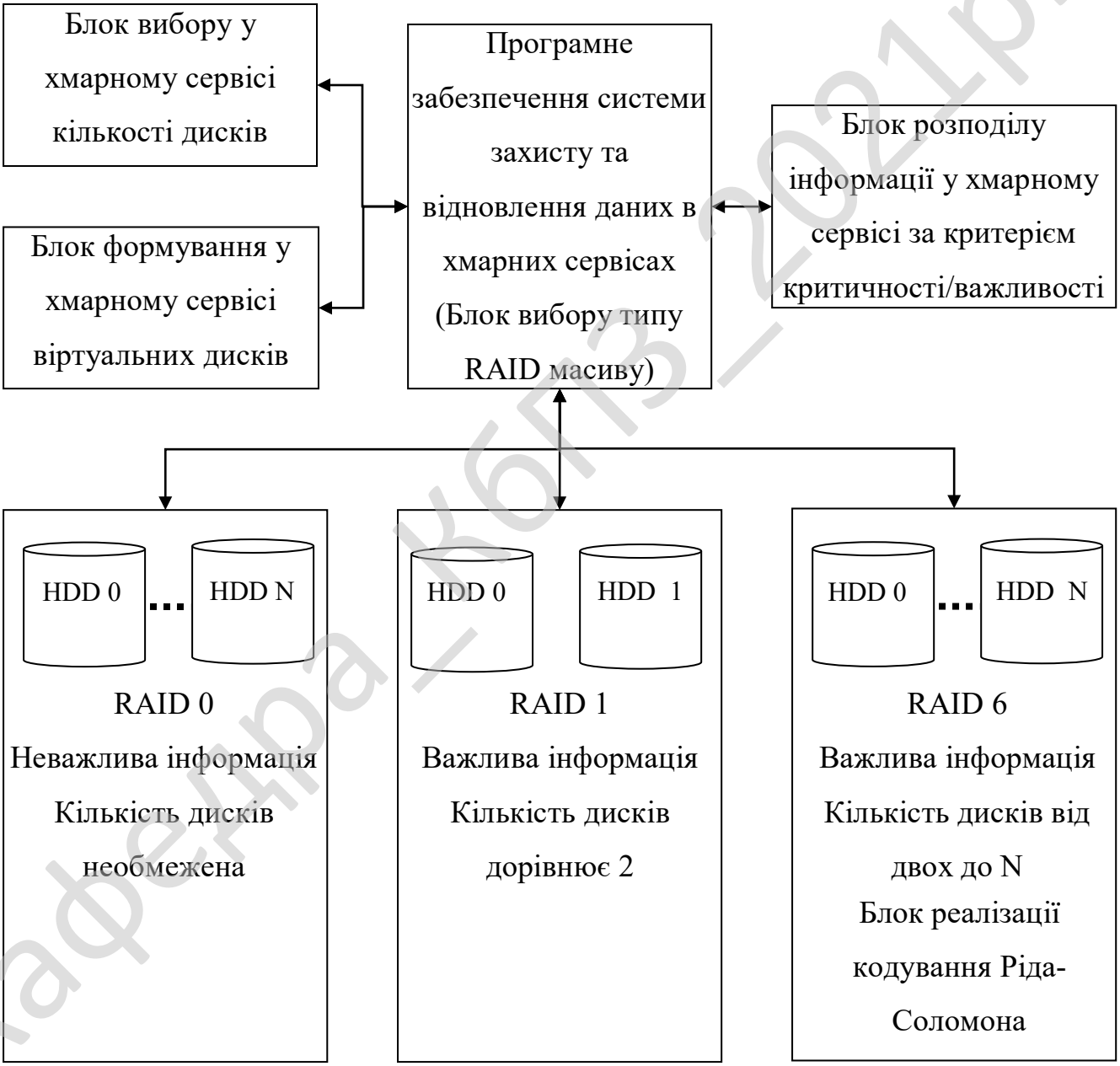


Рисунок 3.2 – Функціональна схема системи

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.

**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-Хоквингхєма (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:

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

**Перевірочна матриця**

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

Тоді:

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

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

Тоді нормований поліном $g(x)$ мінімального ступеня над полем $GF(q)$, корінням якого є $d-1$ підряд, що йдуть ступенів, $\alpha^{l_0}, \alpha^{l_0+1}, \ldots, \alpha^{l_0+d-1}$ елемента $\alpha$, є поліномом, що породжує, коду над *полем* GF(q) $g(x) = (x-\alpha^{l_0})(x-\alpha^{l_0+1})\ldots(x-\alpha^{l_0+d-1})$:, 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 $m(x)$, $\deg m(x) \le k-1$, 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 $2\underline{t}$ додаткових перевірочних символів виправляються t помилок (і менш).

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

**Виправлення багаторазових помилок**

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

Ріда-Соломона над полем $GF(q^m)$ з кодовою відстанню $d = 2t + 1$ можна розглядати як $((q^m - 1)m, (q^m - 1 - 2t)m)$-код над полем $GF(q)$, що може виправляти будь-яку комбінацію помилок, зосереджену в t або меншому числі блоків з m символів.

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

**Практична реалізація**

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

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

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

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

Кодування

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

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

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

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

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

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

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

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

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

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

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

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 ЕАВ. 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 ЕАВ, 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 ЕАВ, використовуючи 4 пари "суматор – ЕАВ", 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

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

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

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

Коректність цього алгоритму випливає з наступних двох тверджень:

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

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

**Знаходження корня**

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

**Визначення характеру помилки і її виправлення**

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

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

## 3.4 Розробка діаграми процесів

Розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3. Основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей.

Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників. Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

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.

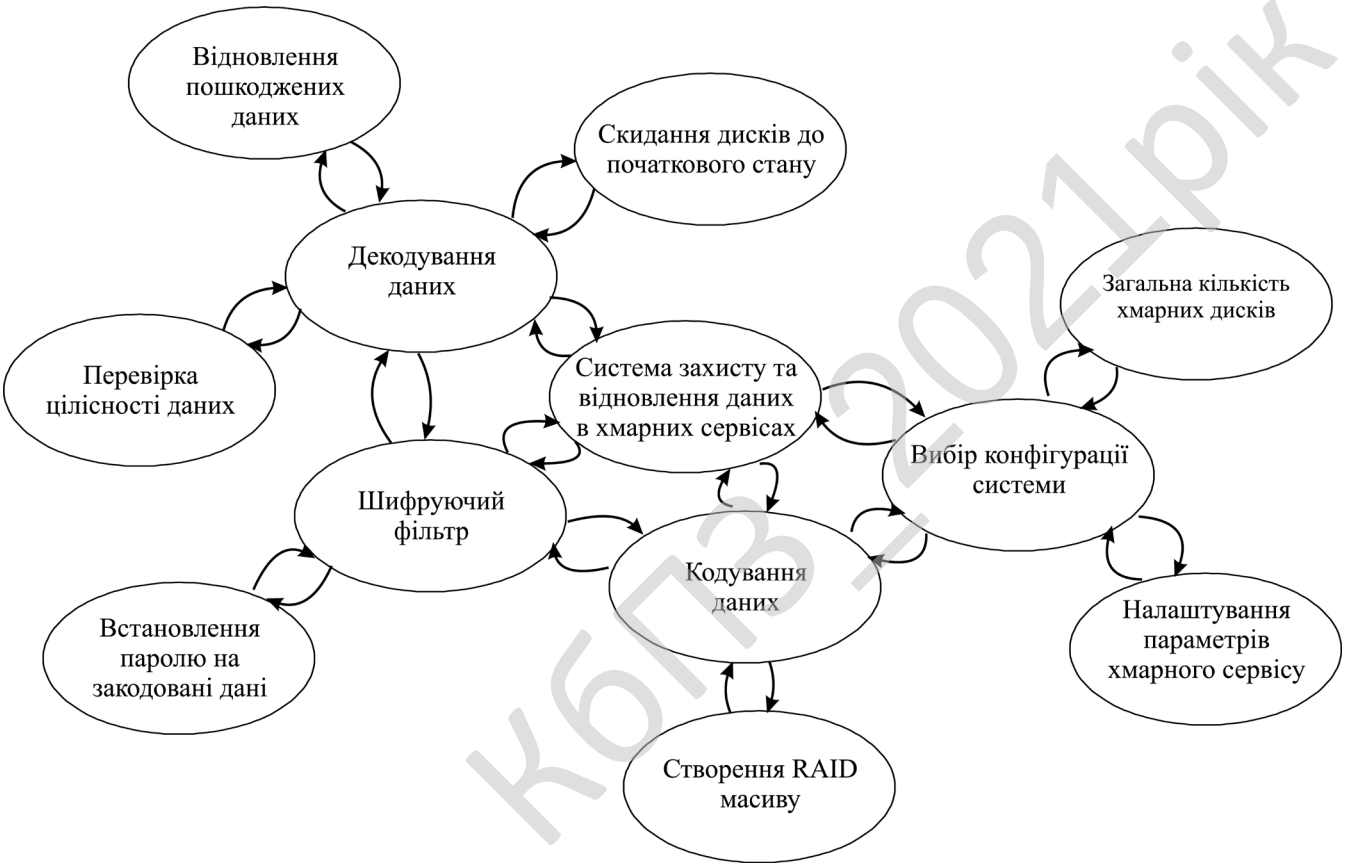


Рисунок 3.3 – Діаграма взаємодії процесів

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграми потоків даних містять чотири типи елементів:

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

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

– Сховища даних (репозиторії).

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

Розглянемо реалізацію магістерської дипломної роботи. Були проведені розрахунки і підібрані набори тестових даних для перевірки правильності реалізації проектних рішень.

Було створено блок-схеми роботи системи. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується. Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо.

Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

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

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

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

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач. Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції. Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу.

Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що ви ходить з вершини (частіше нижній) елемента, яка потім розгалужується.

Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи. Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані. Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи). З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми.

Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

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

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об'єктно-орієнтована що вимагає високого рівня декомпозиції задач на класи.

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

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

Розглянемо визначення API. Це прикладни́й програ́мний інтерфе́йс (Application Programming Interface, API) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

Спрощено – це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб -базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

Одним з найпоширеніших призначень API є надання набору широко використовуваних функцій, наприклад для малювання вікна чи іконок на екрані.

API є абстрактним поняттям – програмне забезпечення, що пропонує деякий API, часто називають реалізацією (implementation) даного API.

У багатьох випадках API є частиною набору розробки програмного забезпечення, водночас, набір розробки може включати як API, так і інші інструменти/апаратне забезпечення, отже ці два терміни не є взаємозамінювані.

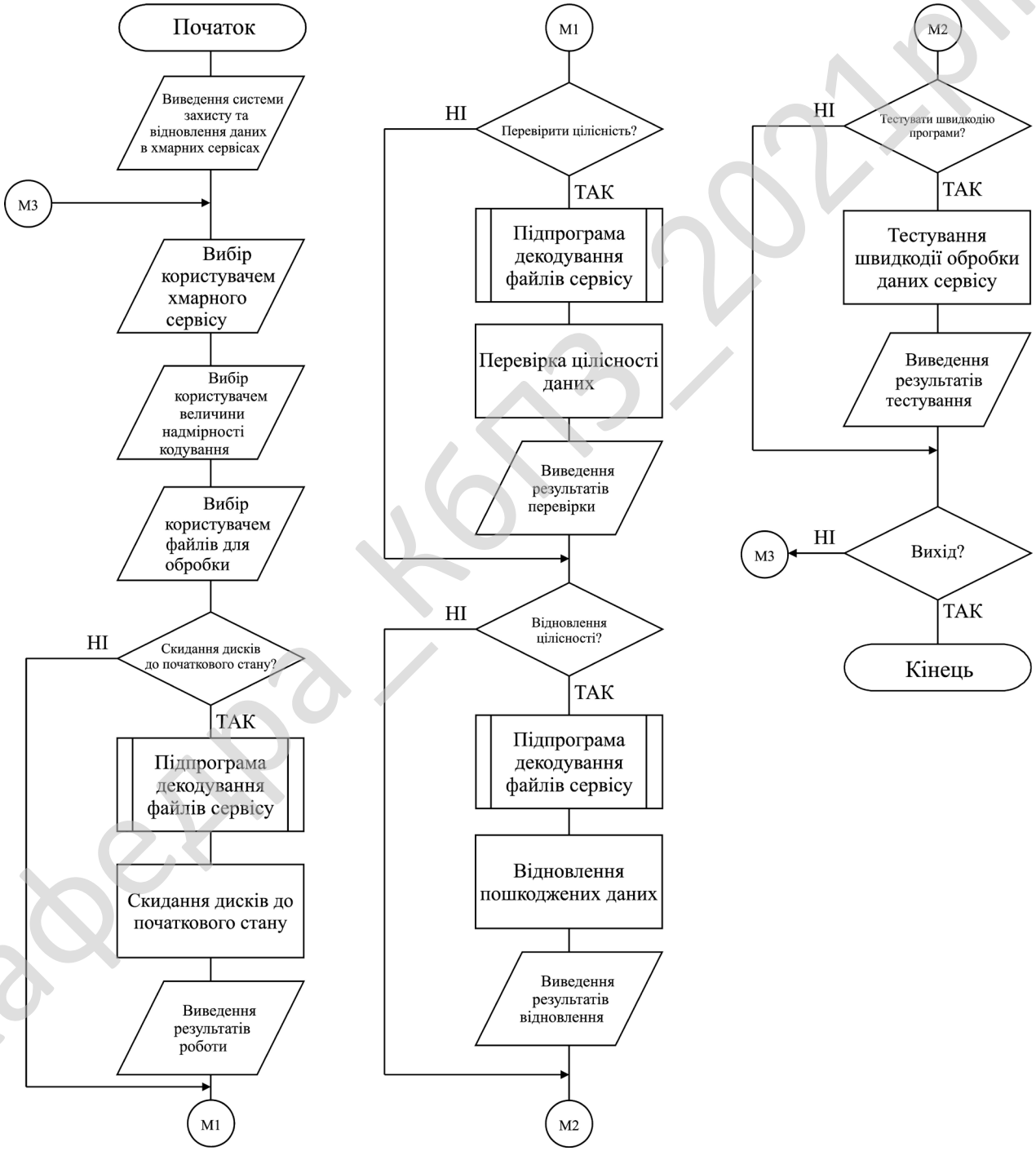


Рисунок 4.1 – Блок-схема основної програми

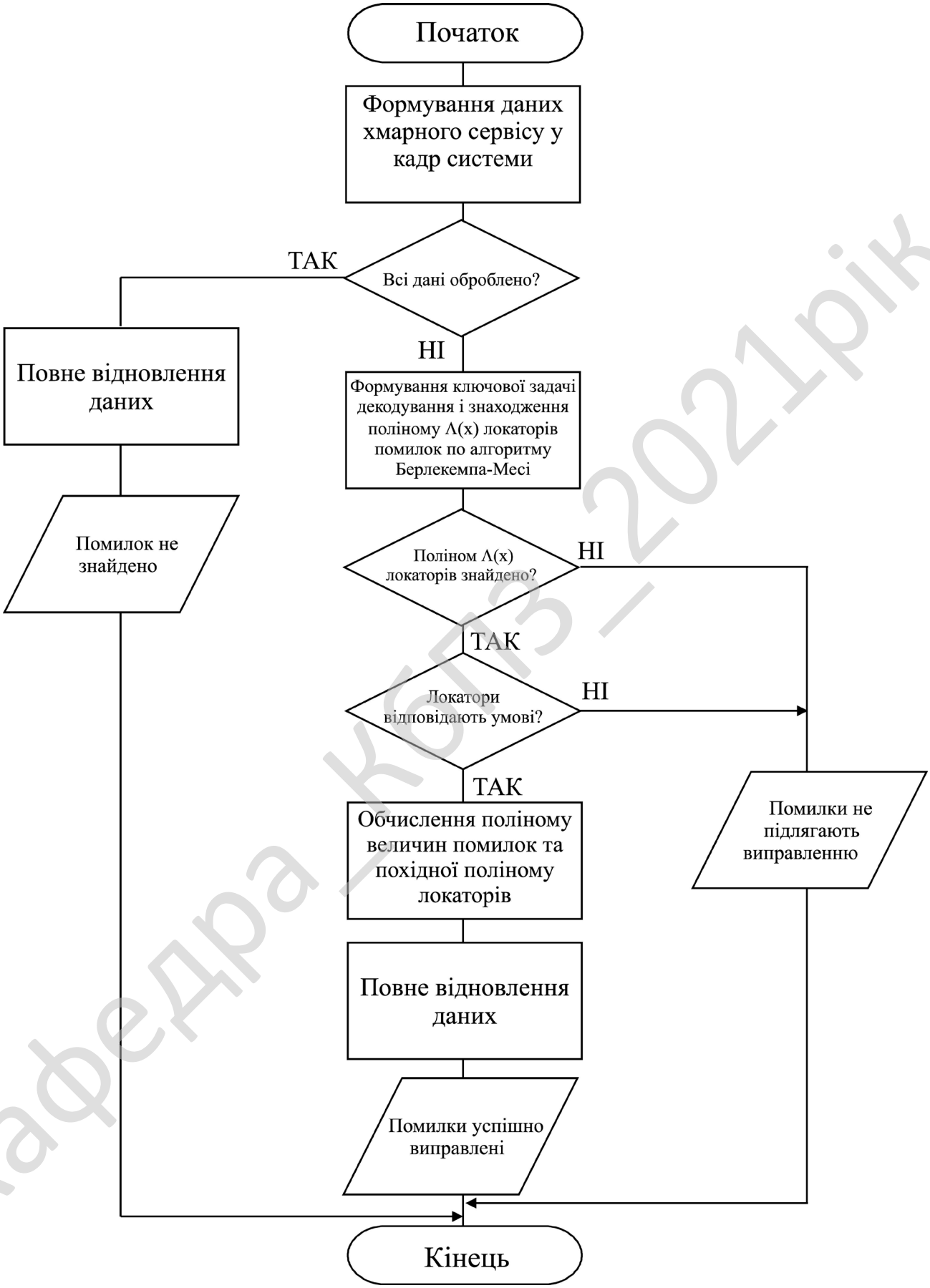


Рисунок 4.2 – Блок-схема роботи підпрограми

Високорівневі API часто програють у гнучкості. Виконання деяких функцій нижчого рівня стає набагато складнішим, або навіть неможливим.

В об'єктно-орієнтованих мовах, прикладний програмний інтерфейс зазвичай включає в себе опис набору визначень класу, з набором форм поведінки, пов'язаних з цими класами. Це абстрактне поняття пов'язане з реальними функціями, які надані або надаватимуться, класами, які реалізуються в методах класу.

Прикладний програмний інтерфейс в даному випадку можна розглядати як сукупність всіх методів, які публічно доступні в класах (зазвичай званий інтерфейс класу). Це означає, що прикладний програмний інтерфейс вказує методи, за допомогою яких взаємодіє з об'єктами, отриманими з визначень класів і обробляє їх.

У більш загальному плані можна визначити Прикладний Програмний Інтерфейс як сукупність усіх видів об'єктів, які можна вивести з визначення класу, і пов'язаних з ними можливих варіантів поведінки.

Наприклад: клас, що представляє Stack, може просто виставити публічно два методи Push() (для додавання нового елемента в стек ) і Pop() (для вилученн я останнього пункту, ідеально розташований на вершині стека).

У цьому випадку Прикладний Програмний Інтерфейс може бути інтерпретованим як два методи pop() і push(), або, більш широко, використовується варіант, коли можна використовувати елемент типу Stack, який реалізує поведінку стека, надаючи йому можливість для додавання / видалення елементів з вершини. Друга інтерпретація видається більш доречною в дусі об'єктно-орієнтованого підходу.

Якість документації, пов'язаної з Прикладним Програмним Інтерфейсом, є часто ключовим фактором, що визначає його успішність з точки зору простоти використання.

ППІ, як правило, пов'язаний із бібліотеками програмного забезпечення: ППІ описує і вказує очікувану поведінку в той час, як бібліотека є фактичною

реалізацією даного набору правил. Один ППІ може мати декілька реалізацій (або жодної, будучи абстрактним) у вигляді різних бібліотек, які мають такий же інтерфейс.

Прикладний програмний інтерфейс також може бути пов'язаним з платформами програмування: платформа може бути заснована на кількох бібліотеках реалізує декілька інтерфейсів ППІ, але на відміну від звичайного використання ППІ, доступ до поведінки вбудований в платформу опосередкований шляхом розширення його змісту новими класами і вставлений в саму платформу. Крім того, загальний потік управління програми може бути під контролем абонента. Прикладний програмний інтерфейс може бути також реалізацією протоколу. Коли ППІ реалізує протокол, він може бути заснованим на проксі-методах віддалених викликів, що засновані на протоколі зв'язку. Роль ППІ може заключатися саме в тому, щоб приховати деталі транспортного протоколу. Наприклад: RMI є ППІ, який реалізує протокол або JRMP IIOP як RMI-IIOP.

Протоколи, як правило, розподіляються між різними технологіями і зазвичай дозволяють різним технологіям обмінюватися інформацією, діючи як абстракція між двома світами. ППІ, як правило, є специфічним для конкретної технології: звідси, інтерфейси даної мови не можуть бути використані на інших мовах, якщо виклики функції не будуть перетво рені з конкретної адаптації бібліотеки.

Деякі мови, серед яких такі, що працюють на віртуальних машинах (наприклад: мови, сумісні з NET CLI середовища CLR і JVM сумісних мов у віртуальній машині Java) можуть ділитися програмними інтерфейсами.

У цьому випадку віртуальна машина дозволяє мові взаємодії завдяки спільному знаменнику віртуальної машини, що абстрагується від конкретної мови, використовувати проміжний байт-код і його мову.

При використанні прикладного програмного інтерфейсу в контексті веб-розробки, як правило, ППІ визначається набором повідомлень запиту HTTP, також визначається структура повідомлень-відповідей, зазвичай у розширенні

мови розмітки XML або в форматі об'єктного запису JavaScript (JSON). У той час як прикладний програмний інтерфейс у Web історично був практично синонімом для веб-служби, останнім часом тенденція змінилась (так званий Web 2.0) на відхід від Simple Object Access Protocol (SOAP) на основі веб-сервісів і сервіс-орієнтованої архітектури (SOA) на більш прямі передачі репрезентативного стану (REST) стилів веб-ресурсів та ресурсів-орієнтованої архітектури (ROA).

Частина цієї тенденції пов'язана з рухом Семантичного веб-ресурсу до Опису Платформ (RDF), Концепції розвитку веб-технологій інженерних онтологій. Прикладні програмні інтерфейси у Web, що дозволяють комбінувати декількома прикладними програмними інтерфейсами в нові додатки називають гібридними.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних програм і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

– набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;

– набір клієнтів, які використовують сервіси, що надаються серверами;

– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою

відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери – це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми – і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

– рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;

– прикладний рівень, який реалізує основну логіку ПЗ і на якому здійснюється необхідна обробка інформації;

– рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

– модель тонкого клієнта, в рамках якої вся логіка ПЗ та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;

– модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Типовим прикладом клієнт-серверної взаємодії є WWW. Існує величезна кількість веб-серверів, на яких розміщується та чи інша інформація. У

найпростішому випадку ця інформація являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML. Але ситуація, як правило, є складнішою; значна частина веб-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

Для того, щоб людина, яка працює в Інтернеті, могла переглянути ту чи іншу сторінку, на її комп'ютері повинно бути встановлено відповідне програмне забезпечення. Програми для перегляду веб-сторінок називаються браузерами.

Але, крім браузерів, до серверів можуть звертатися і інші клієнти, а саме – автономні програми. Вони можуть передбачати взаємодію з людиною, а можуть працювати в цілком автоматичному режимі. Типовим класом таких програм є роботи, призначені для автоматичного перегляду веб-ресурсів. Зокрема, роботи є важливим елементом пошукових систем і використовуються ними для перегляду сторінок і збору інформації про них.

Для запиту до веб-сервера клієнтська програма повинна задати місцезнаходження комп'ютера, на якому розміщується серверна програма, назву потрібного документа і, можливо, інші дані, які специфікують запит. Мережа забезпечує знаходження сервера і передачу йому клієнтського запиту. Серверні програми обробляють цей запит, відповідь пересилається по мережі клієнтові.

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка ПЗ. Програми проміжного рівня можуть функціонувати під управлінням спеціальних серверів ПЗ, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

Для роботи з системою користувач використовує стандартне програмне забезпечення – звичайний браузер. Це позбавляє його необхідності завантажувати

та інсталювати спеціальні програми (хоча інколи така необхідність все-таки виникає).

Але користувачеві слід надати в розпорядженні інтерфейс, який дозволяв би йому взаємодіяти з системою і формувати запити до неї. Форми, що визначають цей інтерфейс, розміщуються на веб-сторінках та завантажуються разом з ними.

Веб-оглядач формує запит та пересилає його до сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних. Сервер даних здійснює операції з даними, що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних.

Найчастіше веб-сервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоч і являють собою окремі і логічно незалежні програмні модулі.

На сучасному етапі для програмування модулів проміжного рівня використовується мова серверних сценаріїв PHP, а для управління даними – СУБД MySQL. Таким чином, зв'язку PHP-MySQL слід розглядати як стандартний інструмент для створення порівняно простих інтерактивних веб-сайтів та систем електронної комерції; близько 90% комерційних систем сьогодні створюється саме на цій основі. Водночас як засоби управління даними, так і middleware-засоби можуть бути найрізноманітнішими. Так, для створення серверних програм, крім PHP, широко застосовуються Java, Perl, Python, Delphi.

Взагалі, технології створення розподілених, зокрема веб-програм, стрімко розвиваються. Слід згадати про технології EJB (Enterprise Java Beans), CORBA, а також про .NET – порівняно нову ініціативу компанії Microsoft. Для зберігання даних та їх передачі часто використовується так звана розширювана мова

розмітки XML (Extensible Markup Language).

При розробці ПЗ було використано V-Model (або VEE модель) є моделлю розробки інформаційних систем (ІС), спрямованої на спрощення розуміння складнощів, пов'язаних з розробкою систем. Вона використовується для визначення єдиної процедури розробки програмного забезпечення, апаратного забезпечення та людино-машинного інтерфейсу.

Концепція V-подібної моделі була розроблена Німеччиною та США в кінці 1980-х років незалежно один від одного:

– Німецька V-модель була розроблена аерокосмічної компанією IABG в Оттобрунні поряд з Мюнхеном у сприянні з Федеральним департаментом з закупівлі озброєнь в Кобленці, для Міністерства оборони Німеччини. Модель була прийнята німецькою федеральною адміністрацією для цивільних потреб влітку 1992.

– Американська V-Model (VEE) була розроблена національною радою з системної інженерії (міжнародна – з 1995 року) для супутникових систем, включаючи обладнання, програмне забезпечення та взаємодію з користувачами.

Сучасною версією V-Model є V-Model XT, яка була затверджена в лютому 2005 року. V-модель використовується для управління процесом розробки програмного забезпечення для німецької федеральної адміністрації.

Зараз вона є стандартом для німецьких урядових і оборонних проектів, а також для виробників ПЗ в Німеччині. V-Model являє собою скоріше набір стандартів у галузі проектів, що стосуються розробки нових продуктів. Ця модель багато в чому схожа з Prince2 і описує методи як для проектного управління, так і для системного розвитку.

**Основні принципи**

Основний принцип V-подібної моделі полягає в тому, що деталізація проекту зростає при русі зліва направо, одночасно з плином часу, і ні те, ні інше не може повернути назад. Ітерації в проекті виробляються по горизонталі, між лівою і правою сторонами літери.

Стосовно до розробки інформаційних систем V-Model – варіація каскадної моделі, в якій завдання розробки йдуть зверху вниз по лівій стороні букви V, а завдання тестування – вгору по правій стороні букви V. Усередині V проводяться горизонтальні лінії, що показують, як результати кожної з фаз розробки впливають на розвиток системи тестування на кожній із фаз тестування.

Модель базується на тому, що приймально-здавальні випробування ґрунтуються, насамперед, на вимогах, системне тестування – на вимогах та архітектури, комплексне тестування – на вимогах, архітектурі та інтерфейсах, а компонентне тестування – на вимогах, архітектурі, інтерфейсах та алгоритмах

**Цілі**

V-модель забезпечує підтримку у плануванні та реалізації проекту. В ході проекту ставляться такі завдання:

1. Мінімізація ризиків: V-подібна модель робить проект більш прозорим і підвищує якість контролю проекту шляхом стандартизації проміжних цілей і опису відповідних їм результатів та відповідальних осіб. Це дозволяє виявляти відхилення в проекті і ризики на ранніх стадіях і покращує якість управління проектом.

2. Підвищення та гарантії якості: V-Model – стандартизована модель розробки, що дозволяє домогтися від проекту результатів бажаної якості. Проміжні результати можуть бути перевірені на ранніх стадіях. Універсальне документування полегшує читаність, зрозумілість та контрольованість.

3. Зменшення загальної вартості проекту: Ресурси на розробку, виробництво, управління і підтримку можуть бути заздалегідь прораховані та проконтрольовані. Отримувані результати також універсальні і легко прогнозуються. Це зменшує витрати на подальші стадії та проекти.

4. Підвищення якості комунікації між учасниками проекту: Універсальний опис усіх елементів та умов полегшує взаєморозуміння всіх учасників проекту. Таким чином, зменшуються неточності у розумінні між користувачем, покупцем, постачальником і розробником.

Переваги:

– Користувачі V-Model беруть участь у розробці та підтримці V-моделі. Комітет з контролю за змінами підтримує проект і збирається раз на рік для обробки всіх отриманих запитів на внесення змін до V-Model.

– На старті будь-якого проекту V-подібна модель може бути адаптована під цей проект, так як ця модель не залежить від типів організацій та проектів.

– V-model дозволяє розбити діяльність на окремі кроки, кожен з яких буде включати в себе необхідні для нього дії, інструкції до них, рекомендації та докладне пояснення діяльності.

Обмеження. Наступні моменти не враховуються в V-моделі, але можуть бути розглянуті окремо, або можливо адаптувати модель під них:

– Не регулюється розміщення контрактів на обслуговування.

– Організація і виконання управління, обслуговування, ремонту та утилізації системи не враховуються в V-моделі. Однак, планування і підготовка до цих операцій моделлю розглядаються.

– V-подібна модель більше стосується розробки програмного забезпечення в проекті, ніж всієї організації процесу.

Переваги:

– У моделі особливе значення надається плануванню, спрямованому на верифікацію та атестацію розроблювального продукту на ранніх стадіях його розробки. Фаза модульного тестування підтверджує правильність деталізованого проектування. Фази інтеграції та тестування реалізують архітектурне проектування або проектування на вищому рівні. Фаза тестування системи підтверджує правильність виконання етапу вимог до продукту і його специфікації.

– У моделі передбачені атестація та верифікація всіх зовнішніх і внутрішніх отриманих даних, а не тільки самого програмного продукту.

– У V-подібної моделі визначення вимог виконується перед розробкою проекту системи, а проектування ПО – перед розробкою компонентів.

– Модель визначає продукти, які повинні бути отримані в результаті процесу розробки, причому кожні отримані дані повинні піддаватися тестуванню.

– Завдяки моделі менеджери проекту можуть відслідковувати хід процесу розробки, так як в даному випадку цілком можливо скористатися тимчасовою шкалою, а завершення кожної фази є контрольною точкою.

**Недоліки:**

– Модель не передбачає роботу з паралельними подіями.

– У моделі не передбачено внесення вимоги динамічних змін на різних етапах життєвого циклу.

– Тестування вимог в життєвому циклі відбувається занадто пізно, внаслідок чого неможливо внести змін, не вплинувши при цьому на графік виконання проекту.

– У модель не входять дії, спрямовані на аналіз ризиків.

– Деякий результат можна отримати тільки при досягненні низу букви V.

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

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

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

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

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

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

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

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

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

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

Є комерційним продуктом, який може бути ліцензований для роботи на локальному сервері або доступний в якості віддаленого додатки. Ціноутворення

залежить від максимального числа користувачів, при цьому близько $50 за користувача для локального і $7 на місяць за користувача для віддаленого доступу є типовими цінами.

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

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

– проект використовує ліцензії, схвалені Open Source Initiative;

– Вихідний код проекту доступний для скачування;

– у проекту є публічно доступна веб-сайт;

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

### 4.2 Захист розробленого програмного забезпечення

Дані у системі захищаються за допомогою алгоритму обміну ключа Діффі-Хеллмана.

Ціль алгоритму полягає в тому, щоб два учасники могли безпечно обмінятися ключем, що надалі може використовуватися в якому-небудь алгоритмі симетричного шифрування. Сам алгоритм Діффі-Хеллмана може застосовуватися тільки для обміну ключами.

Алгоритм заснований на труднощі обчислень дискретних логарифмів. Дискретний логарифм визначається в такий спосіб. Уводиться поняття примітивного кореня простого числа Q як числа, чиї ступені створюють всі цілі від 1 до Q – 1. Це означає, що якщо A є примітивним коренем простого числа Q, тоді числа:

$$A \bmod Q, A^2 \bmod Q, \ldots, A^{Q-1} \bmod Q,$$

є різними й складаються із цілих від 1 до Q – 1 з деякими перестановками. У цьому випадку для будь-якого цілого $Y < Q$ і примітивного кореня A простого числа Q можна знайти єдину експоненту X, таку, що:

$$Y = A^X \bmod Q,$$

де $0 \le X \le (Q - 1)$.

Експонента X називається дискретним логарифмом, або індексом Y, по підставі A mod Q. Це позначається як $\text{ind}_{, Q}(Y)$.

Тепер опишемо алгоритм обміну ключів Діффі-Хеллмана.

**Загальновідомі елементи**

Q – просте число.

A – A < Q і A є примітивним коренем Q.

**Створення пари ключів користувачем I**

Вибір випадкового числа $X_i$ (закритий ключ):

$$X_i < Q.$$

Обчислення числа $Y_i$ (відкритий ключ):

$$Y_i = A^{Xi} \bmod Q.$$

**Створення відкритого ключа користувачем J**

Вибір випадкового числа $X_j$ (закритий ключ):

$$X_j < Q.$$

Обчислення випадкового числа $Y_j$ (відкритий ключ):

$$Y_j = A^{Xj} \bmod Q.$$

**Створення загального секретного ключа користувачем I**

$$K = (Y_j)^{Xi} \bmod Q.$$

**Створення загального секретного ключа користувачем J**

$$K = (Y_i)^{Xj} \bmod Q.$$

Передбачається, що існують два відомих усім числа: просте число Q і ціле A, що є примітивним коренем Q.

Тепер припустимо, що користувачі I і J хочуть обмінятися ключем для алгоритму симетричного шифрування.

Користувач I вибирає випадкове число $X_i < Q$ і обчислює:

$$Y_i = A^{Xi} \bmod Q.$$

Аналогічно користувач J незалежно вибирає випадкове ціле число $X_j < Q$ і обчислює:

$$Y_j = A^{Xj} \bmod Q.$$

Кожна сторона тримає значення X у секреті й робить значення Y доступним для іншої сторони.

Тепер користувач I обчислює ключ як $K = (Y_j)^{Xi} \bmod Q$, і користувач J обчислює ключ як $K = (Y_i)^{Xj} \bmod Q$. У результаті обоє одержать те саме значення:

$$K = (Y_j)^{Xi} \bmod Q = (A^{Xj} \bmod Q)^{Xi} \bmod Q = (A^{Xj})^{Xi} \bmod Q =$$

за правилами модульної арифметики

$$= A^{Xj\,Xi} \bmod Q = (A^{Xj})^{Xj} \bmod Q = (A^{Xi} \bmod Q)^{Xj} \bmod Q = (Y_i)^{Xj} \bmod Q$$

Таким чином, дві сторони обмінялися секретним ключем. Так як $X_i$ і $X_j$ є закритими, супротивник може одержати тільки наступні значення: Q, A, $Y_i$ і $Y_j$. Для обчислення ключа атакуючий повинен зламати дискретний логарифм, тобто обчислити:

$$X_j = \mathrm{ind}_{a,\,q}(Y_j).$$

Безпека обміну ключа в алгоритмі Діффі-Хеллмана випливає з того факту, що, хоча відносно легко обчислити експоненти по модулю простого числа, дуже важко обчислити дискретні логарифми. Для великих простих чисел задача вважається нерозв'язною.

Варто помітити, що даний алгоритм уразливий для атак типу "in-the-middle". Якщо супротивник може здійснити активну атаку, тобто має можливість не тільки перехоплювати повідомлення, але й заміняти їх іншими, він може перехопити відкриті ключі учасників $Y_i$ і $Y_j$ , створити свою пару відкритого й закритого ключа ($X_{оп}$, $Y_{оп}$) і послати кожному з учасників свій відкритий ключ. Після цього кожний учасник обчислить ключ, що буде загальним із супротивником, а не з іншим учасником. Якщо немає контролю цілісності, то учасники не зможуть виявити подібну підміну.

# 5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено розроблене у магістерської дипломної роботі система захисту та відновлення даних в хмарних сервісах. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи: – Функціональних кнопок ПЗ; Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші; Розділу обрання групи; Розділу виведення результату роботи системи; Функції представлені у графічному вигляді.

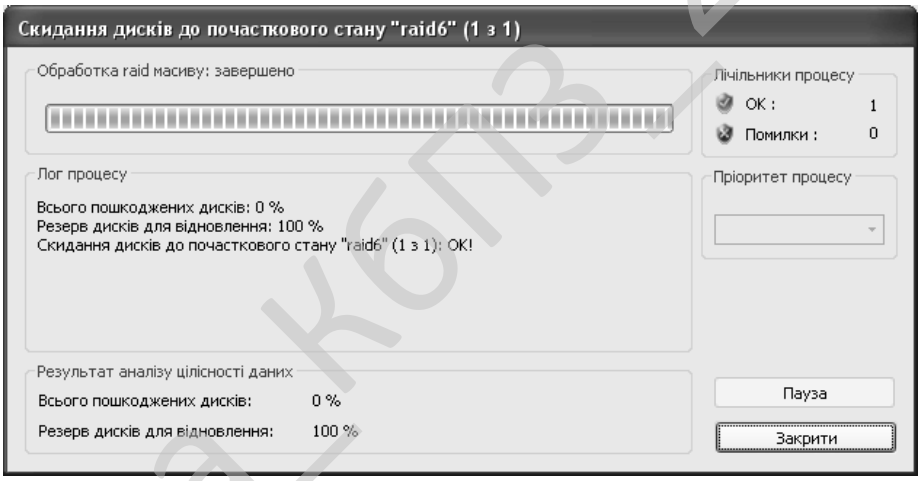


Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і 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.

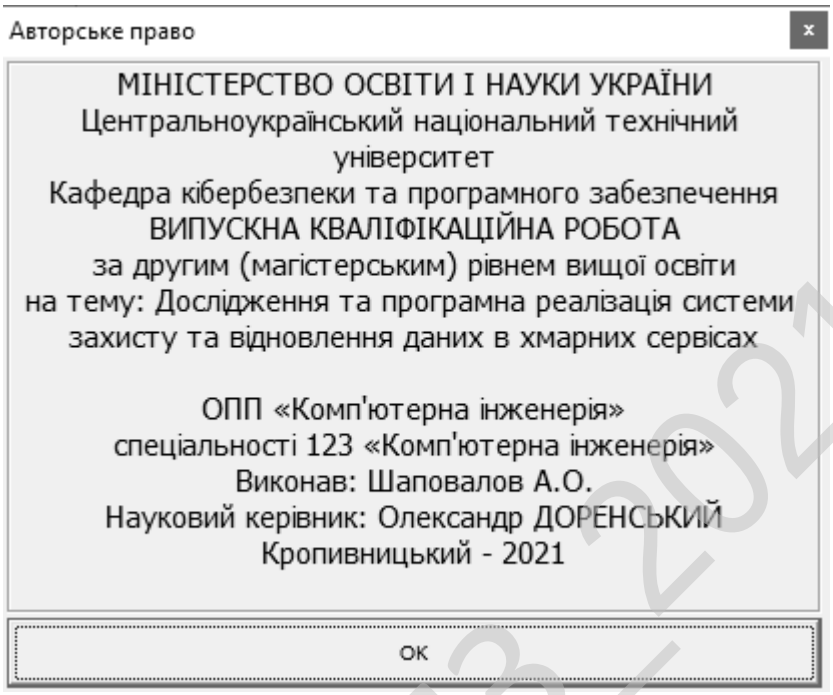


Рисунок 5.2 – Авторське право

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 ІТ рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене ІТ рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження.

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

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути

невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

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

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

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

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

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

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

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

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

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

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

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

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

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

Будь-який спосіб тестування «чорної скриньки» повинен:

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

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

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

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій;

– Помилок інтерфейсу;

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

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

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Freeware. Це власницьке програмне забезпечення, котре можна Безоплатно використовувати протягом необмеженого терміну без обмежень у функціональності, і поширюване без сирцевих кодів.

Автори такого програмного забезпечення, як правило, хочуть «дати щось спільноті», але хочуть також контролювати його подальшу розробку. Іноді, коли програмісти вирішують припинити розробку, вони передають сирцевий код іншим програмістам, або ж спільності як вільне програмне забезпечення.

Дуже часто плутають поняття «безплатне програмне забезпечення» та «вільне програмне забезпечення», хоча вони суттєво відрізняються.

Безплатне програмне забезпечення можна безоплатно встановлювати та використовувати (іноді з певними обмеженнями, як, наприклад, «безплатне для домашнього або некомерційного вжитку»), в той час як вільне програмне забезпечення можна продавати за будь-яку суму, але при тому, у користувача, котрий його отримує, повинні бути права на вивчення, модифікацію та поширення сирцевих кодів одержаної програми.

# 6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи захисту та відновлення даних в хмарних сервісах.

*Метою розробки* є дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах.

*Об'єктом дослідження* є процес захисту та відновлення даних в хмарних сервісах.

*Предметом дослідження* є методи захисту та відновлення даних в хмарних сервісах.

*Методи дослідження* базуються на методах хмарних технологій, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод захисту та відновлення даних в хмарних сервісах.

– Розроблено вітчизняний продукт захисту та відновлення даних в хмарних сервісах, який має більш широкі можливості, на відміну від існуючих аналогів.

# 7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

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

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці). В магістерській роботі було проведене дослідження та виконана програмна реалізація системи захисту та відновлення даних в хмарних сервісах.

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

а) невеликий розмір;

б) невеликі системні потреби;

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

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

| Показники | Позначення | Характеристика або величина |
|---|---|---|
| 1 | 2 | 3 |
| 1. Кількість розроблених програм період, шт. | N | 1 |
| 2. Кількість екземплярів програм, шт. | Ne | 280 (2 ост. цифри № зал $*10^1$) |
| 3. Запланований термін розробки, днів | Fpq | 60 (3 місяці) |
| 4. Група задачі підсистеми управління (1-6) | – | 1 |
| 5. Ступінь новизни задачі (А, Б, В, Г) | – | Б |
| 6. Складність алгоритму (1, 2, 3) | – | 2 |

Продовження таблиці 7.1

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

Продовження таблиці 7.1

| 1 | 2 | 3 |
|---|---|---|
| 28. Досвід роботи з мовою і інструментами середовища розробки (1-6) | – | 2 |
| 29. Досвід роботи з програмними інструментами розробки (1-6) | – | 3 |
| 30. Розробка ПЗ для декількох серверів одночасно (1-6) | – | 2 |
| 31. Вимоги до дотримання встановленого графіка робіт (1-6) | – | 2 |
| 32. Вартість ПЗ у розробника (НМА), грн. | – | 28000 (2 ост. цифри № за $*10^3$) |
| 33. Норматив додаткової зарплати, % : | Нд | 10 |
| 34. Норматив відрахувань у соціальні фонди, % | Нс | 22 |
| 35. Норматив загальногосподарських витрат, % | Нг | 15 |
| 36. Норматив витрат на освоєння нових мов програмування, % | Нп | 15 |
| 37. Рівень рентабельності програмної продукції, % | Ре | 50 |
| 38. Ставка податку на додану вартість, % | Ндв | 20 |

### 7.2 Розрахунок трудомісткості розробки програмної продукції

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

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

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

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

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

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

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

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

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

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

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

$$T_{уточн} = 6{,}78\cdot(0{,}88\cdot 0{,}93\cdot 0{,}88\cdot 0{,}91\cdot 0{,}95\cdot 1\cdot 1\cdot 0{,}87\cdot 1{,}22\cdot 1{,}16\cdot 1{,}1\cdot 1{,}1\cdot$$

$$\cdot 1{,}12\cdot 1{,}1\cdot 1\cdot 1{,}1\cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

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

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

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

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

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

$$T_{РП} = 0{,}3\cdot 3{,}23\cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)}\cdot 82 = 168 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

| Стадії розробки | Трудомісткість за типовими нормами та розрахунками | |
|---|---|---|
| | Величина, люд/дні | Підстава |
| Технічне завдання | 9 | Д5 |
| Ескізний проект | 10 | Д6 |
| Технічний проект | 9 | Д7 |
| Робочий проект | 168 | Ф 7.1-7.4 |
| Впровадження | 13 | Д13 |
| Всього | 209 | – |

### 7.3 Визначення чисельності виконавців і планового фонду зарплати

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

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

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

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

$$Ч = \frac{209 \cdot 1}{60 - 5} = 3,8 \text{ ставки.}$$

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

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

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

| Найменування обладнання | Профілактичне обслуговування | | | |
|---|---|---|---|---|
| | Кількість хв. на один. обл. | Кількість облад-нання | Затрати часу в хв. | Затрати часу в год. |
| Системний блок ПК | 385 | 12 | 4620 | 77 |
| Монітор | 160 | 12 | 1920 | 32 |
| Клавіатура | 140 | 12 | 1680 | 28 |
| Маніпулятор «мишка» | 30 | 12 | 360 | 6 |
| Принтер матричний | 185 | 1 | 185 | 3 |
| Принтер лазерний | 355 | 2 | 710 | 12 |
| Принтер струминний | 300 | 1 | 300 | 5 |
| Сканер | 155 | 2 | 310 | 5 |
| Концентратор-маршрутизатор | 155 | 2 | 310 | 5 |
| Кабельні господарства ЛОМ на 1 м. п. | 2,5 | 100 | 250 | 4 |
| Кабельне господарство електромережі | 48 | 50 | 2400 | 40 |
| Копіювальний апарат | 285 | 2 | 570 | 10 |
| Усього за рік: | | | $З_ч$ | 227 |

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

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

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

$$Ф_{др}^{с} = \frac{227 \cdot 3}{1{,}2} = 567{,}5 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

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

$$Ч_{ел} = 567{,}5/(60 \cdot 8) = 1{,}2 \text{ ставки.}$$

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

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

| Посада | Вид роботи | Час | К-ть штатних одиниць |
|---|---|---|---|
| Адміністратор загальної мережі, аналітик | Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2019, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN PPPoE, Frame Relay, Wi-Fi | 2 | 0,5 |
| | Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS) | 0,5 | |
| | Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ | 0,5 | |
| | Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет | 1 | |
| Всього | | 4 | |

Продовження таблиці 7.4

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

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

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

| Посада | Кількість ставок | Середньомісячний оклад, грн. | Всього за період розробки, грн. |
|---|---|---|---|
| Керівник (ІТ-менеджер) | 1 | 12000 | 36000 |
| Продакт-менеджер | 0,25 | 8000 | 6000 |
| Інженер-програміст | 3,8 | 8000 | 91200 |
| Інженер-електронщик | 1,2 | 6000 | 21600 |
| Інженер-системотехнік | 0,25 | 6000 | 4500 |
| Адміністратор мережі | 0,5 | 6000 | 9000 |
| Системний програміст | 0,25 | 6000 | 4500 |
| Дизайнер WEB | 0,25 | 8000 | 6000 |
| Інженер-верстальник | 0,25 | 6000 | 4500 |
| Бухгалтер-економіст | 0,5 | 10000 | 15000 |
| Всього за період розробки | $R_{сп}$ = 8,25 | - | $Ф_{роб}$ = 198300 |

Розрахуємо середньоденну зарплату одного виконавця:

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

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

$$З_{сд} = \frac{198300}{8,25 \cdot 60} = 401 \; грн.$$

## 7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

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

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

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

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

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

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пекторaль» (м. Кіровоград) ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 800…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 25 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м$^2$. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

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

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

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

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

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

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

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

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу фірми Компбест за 06.10.21 – джерело https://compbest.com.ua/.

Таблиця 7.6 – Специфікація

| Найменування комплектуючої або обладнання | Тип | Оптова ціна |
|---|---|---|
| Персональний комп’ютер | | 10947 |
| Системний блок | | 7347 |
| Процесор | Intel Core i5-4590 (4 ядра по 3.3-3.7GHz), MB cache | 1750 |
| Системна плата | MSI H81M-P33 6 x USB 2.0, 2 x USB 3.( VGA, DVI, 2 x PS/2, LAN (RJ-45), 5 x Audi FireWire | 1200 |
| Відеокарта | nVidia GeForce GTX 660, 2 GB G DDR5, 192 bit | 750 |
| Жорсткий диск | 240 GB SSD | 1200 |
| Оперативна пам’ять | Samsung DDR3-1333 8Gb PC3-10600R EC Registered (M393B1K70CH0-CH9Q5) | 900 |
| DVD-привод | DVDRW Pioneer DVR-TD10RS SATA Slir Black Bulk (DVR-TD10RS) | 416 |
| Корпус | ATX Middle Tower FOXCONN Pro, 3GTLA 489, PSU 350W(FSP Brand: ATX-350PNR 12cm), black, (front bezel – black+light silve body material – 0.6mm), 80mm fan (rear 2xUSB2.0/AUDIO/MIC, Air Duct, Tool-les chassis design,Thermally Advantaged Chassis | 911 |

Продовження таблиці 7.6

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

Таблиця 7.7 – Балансова вартість обчислювальної техніки

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

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

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

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

## 7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

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

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 401 \cdot 209/280 = 300 \text{ грн.}$$

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

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

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

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

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

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

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

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

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

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

де: $Н_г$ – загальногосподарські витрати, %.

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

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

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

де: $З_{М1}$ – вартість паперу, грн.; $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.; $З_{М3}$ – вартість фарби, картріджей, тонеру, грн.; $N_е$ – кількість екземплярів програм, шт.

Згідно виданих викладачем норм приймаємо одну пачку паперу на три місяці розробки ($n_p$=0,33). Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$ = 105 грн., визначаємо вартість паперу за період розробки $N_м$ = 3 міс:

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

$$З_{М1} = 105 \cdot 0{,}33 \cdot 3 = 105 \text{ грн.}$$

Згідно виданих викладачем норм до вартості запам'ятовуючих пристроїв входить вартість CD дисків в кількості, що дорівнює кількості екземплярів програм та одного DVD диска для збереження резервної копії програми:

$$З_{М2} = \sum Ц_д\ , \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 2 грн./шт., DVD-R LG 4,7Gb, 16x speed Cake box – 2 грн./шт.

$$З_{М2} = 280 \cdot 12 = 3360 \text{ грн.}$$

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

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

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

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

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

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

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \quad (7.19)$$

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

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

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

$$A_м = \frac{A_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

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

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

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

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

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

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

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

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

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

Для даного програмного забезпечення рівень рентабельності складає 50%.

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

де: $P_n$ – рівень рентабельності, %.

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

## 7.6 Визначення об'єму капітальних вкладень у споживача програмної продукції

Об'єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

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

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

## 7.7 Визначення експлуатаційних витрат

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

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

| Найменування статей витрат | Позначення | Сума витрат за варіантами, грн. | |
|---|---|---|---|
| | | Базовий | Новий |
| 1. Витрати на обслуговування системи | $З_р$ | 15190 | 3032 |
| 2. Витрати на електроенергію | $З_{ел}$ | 1488 | 1030 |
| 3. Витрати на амортизацію | $З_{ам}$ | 0 | 307 |
| Всього витрат за рік | $І$ | 16678 | 4369 |

Витрати на профілактичні роботи:

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

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

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

Після купівлі нового програмного забезпечення витрати на обслуговування системи зменшились з 15190 грн до 3032 грн на рік.

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

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

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

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

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

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

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

## 7.8 Визначення економічної ефективності програмної продукції

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

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

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

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

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

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

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

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

$$T_в = \frac{1760367}{(1022 - 681) \cdot 280 \cdot 12 / 3} = 4{,}6 \text{ роки}$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

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

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (16678\text{-}4369)\text{-}0{,}25 \cdot 1226 = 12003 \text{ грн.}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

| Найменування показників | Одиниця виміру | Величина |
|---|---|---|
| 1. Кількість екземплярів програми | Прим. | 280 |
| 2. Повна собівартість розробленої програми | Грн. | 681 |
| 3. Ціна розробленої програми | Грн. | 1022 |
| 4. Плановий прибуток від реалізації розробленої програми | Грн. | 341 |
| 5. Рентабельність програмної продукції | % | 50 |
| 6. Об'єм додаткових капітальних вкладень у виробника програмної продукції | Грн. | 1760367 |
| 7. Загальний прибуток від реалізації програмної продукції | Грн. | 95480 |
| 8. Величина економічного ефекту при виготовлені програмної продукції | Грн. | 50208 |
| 9. Період окупності додаткових капітальних вкладень у виробника програмної продукції | Роки | 4,6 |
| 10.Об'єм додаткових капітальних вкладень у споживача програмної продукції | Грн. | 1226 |
| 11.Величина економічного ефекту у користувача програмної продукції | Грн. | 12003 |
| 12.Період окупності додаткових капітальних вкладень у користувача програмної продукції | Років | 0,1 |

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

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \tag{7.28}$$

$$T_{сп} = \frac{1226}{16678 - 4369} = 0{,}1 \; року \, .$$

### 7.9 Висновки

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

# 8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

## 8.1 Вступ

Законом України “Про охорону праці” [1] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

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

Ці шкідливі фактори можуть привести до професійних захворювань.

При розгляді шкідливих чинників роботи програмістів та інших спеціалістів ІТ будемо керуватись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98, та «Правила охорони праці під час експлуатації електронно-обчислювальних машин» НПАОП 0.00-1.28-10,

Умови праці програміста включають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

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

## 8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Програміст працює з електронно-обчислювальною машиною (ЕОМ) та іншим обладнанням, яке є джерелом небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. Так як програміст постійно перебуває в приміщенні, тому для комфортних умов праці в цьому приміщенні необхідно створити належний мікроклімат.

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

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

– ризик виникнення пожежі;

– негативний вплив на органи зору людини;

– ризики ураження електричним струмом;

– недостатня, або надмірна освітленість робочого місця;

– монотонність праці;

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

– несприятливі мікрокліматичні умови;

– нервово-емоційна напруженість праці;

– інтелектуальні навантаження;

– невідповідність ергономічних показників робочого місця діючим вимогам;

– шуми;

– статичні навантаження на кістково-м’язовий апарат;

## 8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

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

Таблиця 8.1 - Розміри приміщення

| Найменування | Значення, м |
|---|---|
| Ширина | 8,5* |
| Довжина | 13* |
| Висота | 2,9 |

* вказано загальні розміри поєднаного приміщення, де загалом працюють 16 людей, а фактично у наявності є дві кімнати, розділених перестінком.

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 | 6,9 |
| Обсяг, V | $м^3$ | не менше 20.0 | 20 |

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

У зазначеному приміщенні працє 16 осіб. 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 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», ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»).

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, яка виконується в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря у приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

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

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У

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

Таблиця 8.3 - Оптимальні і фактичні значення параметрів мікроклімату

| Пора року | Оптимальні для Іа | | | Фактичні | | |
|---|---|---|---|---|---|---|
| | Температура, °С | Воло-гість,% | Швидкість повітря, м/с | Температура, °С | Воло-гість% | Швидкість повітря, м/с |
| Холодна | 22-24 | 40-60 | 0,1 | 22-23 | 40-55 | 0,1 |
| Тепла | 23-25 | 50-70 | 0,1 | 23-24 | 55-70 | 0,14 |

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 Laser 135a*, 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.

Працю працівника, який постійно працює за комп’ютером, згідно ДБН В.2.5 – 28 – 2006 р. можна віднести до роботи з мал ою 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. Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

### 8.4 Розробка заходів з умов поліпшення охорони праці

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

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

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значень;

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

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

## 8.5 Розрахункова частина

Початкові данні для розрахунку захисного заземлення.

Тип заземлення: робоче заземлення нульової точки трансформатора. Заземленню підлягає виробниче обладнання фірми. Напруга – 220/380 В.

Розрахунок проводиться для випадку розміщення заземлюючого пристрою в однорідній землі за допустимим опором розтіканню струму заземлювача методом коефіцієнта використання заземлювачів.

Для заземлення застосовуються: вертикальні електроди – пруток діаметром $D$=12 мм, довжиною $L$ = 2,5 м. Відстань між вертикальними електродами заземлювача $A$=2,5 м. Розміри перетину з'єднуючої полоси: 40 х 4 мм. Глибина закладення контура заземлення $t$= 0,7 м. Товщина верхнього шару грунта $H$=1,7 м. Ширина (діаметр) горизонтального заземлювача: $b$=40 мм. Відстань від центра вертикального заземлювача до поверхні замлі: $T$=1,95 м. Сезонний кліматичний коефіцієнт вертикального заземлювача: Cv=1,4. Сезонний кліматичний коефіцієнт горизонтального заземлювача: Cv=1,5. Питомий опір верхнього шару грунта (суглинок): $\rho_1$= 100 Ом*м. Питомий опір нижнього шару грунта (супесь): $\rho_2$= 150 Ом*м. Тип вертикального заземлювача: пруток діаметром 12 мм. Тип горизонтального заземлювача (з'єднуючої полоси): полоса. Розташування заземлювачив: по контуру. Опір заземлювача, який нормується: $R_{ЗН}$ = 4 Ом. Коефіцієнт використання вертикального заземлювача $\eta_C$ = 0,55 (табличне значення, залежить від кількості вертикальних стержнів та співвідношення A/L). Коефіцієнт використання горизонтального заземлювача=0,34 (табличне значення, залежить від співвідношення A/L).

Розрахунок захисного заземлення можна автоматизувати за допомогою програми (автор О.В.Оришака), сирцевий код якої опублікован на стр.13-16 [3], або будь якої іншої, наприклад, програми «Електрик» (автор Сумеркін Є.О.), URL: http://rzd2001.narod.ru. Остання була використана мною для автоматизації процесу перевірки розрахунку.

Розрахунок.

Еквівалентний питомий опір:

$$\rho_{екв} = \psi \rho_1 \rho_2 L / [\rho_1 \psi(L-H+t) + \rho_2 \psi(H-t)] = 112{,}9 \text{ Ом.}$$

Опір одного вертикального заземлювача:

$$R_O = [\rho_1/(2\pi * L)] * [(ln)2L/D) + 0{,}5ln[(4T+L)/(4T-L)] = 45{,}77 \text{ Ом.}$$

Опір, який нормується, якщо $\rho_{екв} > 100$ (у нас $\rho_{екв} = 112{,}9 > 100$):

$$R = R_{3H} * \rho_{екв}/100 = 4*112{,}9/100 = 4{,}529 \text{ Ом.}$$

Опір розтіканню горизонтального заземлювача (полоси):

$$R_П = 0{,}366(\rho_{екв} \psi / L_П \cdot \eta_П) \lg(2 / L_П * L_П / bt) = 30{,}03 \text{ Ом.}$$

Де довжина горизонтального заземлювача (полоси):

$L$=A*n =18*2,5=45 м. (при розташуванні заземлювачів по контуру).

Де n – ітераційна кількість вертикальних заземлювачів.

Опір розтіканню вертикальних заземлювачів:

$$R_B = R_П R / (R_П - R) = 4{,}61 \text{ Ом.}$$

Кількість вертикальних заземлювачів:

$$n = R_O / R_B * \eta_C = 18 \text{ шт.}$$

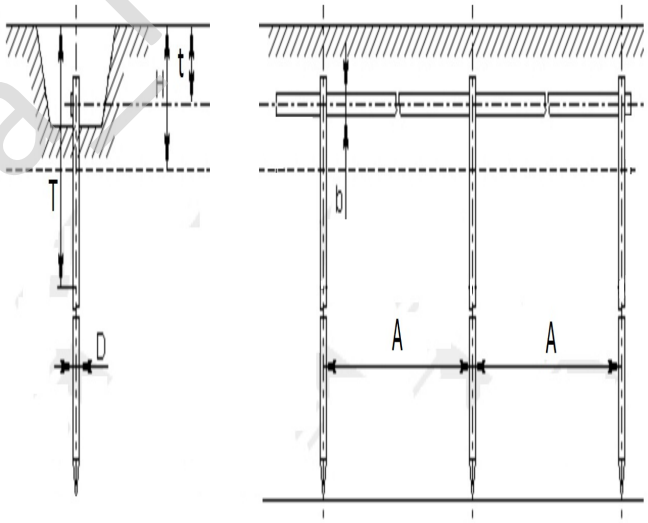


Рисунок 8.1 – Вертикальний електрод заземлювача.

## 8.6 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 штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я 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 захисту та відновлення даних в хмарних сервісах.

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 захисту та відновлення даних в хмарних сервісах.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем захисту та відновлення даних в хмарних сервісах.

– Досліджена система захисту та відновлення даних в хмарних сервісах.

– На основі отриманих результатів досліджень створена програмна реалізація системи захисту та відновлення даних в хмарних сервісах.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання захисту та відновлення даних в хмарних сервісах.

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 XP/Vista/7/8/10.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Діффі-Хеллман.

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

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 12003 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,1 роки.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Шаповалов А.О. Дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах // Збірник праць молодих науковців ЦНТУ. – Вип. 12. – Кропивницький: ЦНТУ, 2022.

2. Коваленко А.С. Разработка структуры базы данных интегрированной информационной системы / А.С. Коваленко, А.В. Коваленко // Информационные технологии и защита информации в информационно-коммуникационных системах: монографія / Под редакцией профессора В.С. Пономаренко. – Х.: Вид-во ТОВ «Щедра садиба плюс», 2015. – С. 54-64.

3. Кожанова А.С. Обґрунтування необхідності створення систем технічної діагностики інтегрованих інформаційних систем / О.А. Смірнов, А.С. Кожанова, О.В. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2013. – Вип. 6(113). – С. 255-257.

4. Коваленко А.С. Задачи распознавания ситуаций в ERP системах / А.В. Коваленко ., А.А. Смірнов, А.С. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2014. – Вип. 4(120). – С. 161-164.

5. Коваленко А.С. Підсистема технічної діагностики для автоматизації процесів керування в інтегрованих інформаційних системах / А.С. Коваленко , О.А.Смірнов, О.В. Коваленко // Системи озброєння і військова техніка.– Х.: ХУПС, 2014. – № 1(37). – С. 126-129.

6. Коваленко А.С. Анализ эффективности использования экспертной системы технической диагностики с традиционной структурой / А.С. Коваленко, А.А. Смирнов, А.В. Коваленко // Системи озброєння і військова техніка. – Х.: ХУПС, 2014. – № 2(38). – С. 106-108.

7. Коваленко А.С. Разработка структуры экспертной системы технической диагностики интегрированной информационной системы /

А.С. Коваленко, А.А. Смирнов, А.В. Коваленко // Наука і техніка Повітряних Сил Збройних Сил України. – Харків: ХУПС, 2014. – № 2(15). – С.154-157.

8. Коваленко А.С. Разработка структуры экспертной системы технической диагностики интегрированной информационной системы / А.С. Коваленко, А.А. Смирнов, А.В. Коваленко // Наука і техніка Повітряних Сил Збройних Сил України. – Харків: ХУПС, 2014. – № 2(15). – С.154-157.

9. Коваленко А.С. Структура системи технічної діагностики інтегрованих інформаційних систем / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Збірник наукових праць Кіровоградського національного технічного університету / техніка в сільськогосподарському виробництві, галузеве машинобудування, автоматизація. – Кіровоград: Вид-во КНТУ, 2014. – Вип. 27. – С. 245-251.

10. Коваленко А.С. Дослідження будови інтегрованої інформаційної системи та її елементів / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Системи озброєння і військова техніка. – Х.: ХУПС, 2014. – № 4(40). – С. 85-88.

11. Коваленко А.С. Розробка структури бази даних для обліку технічного стану елементів інтегрованої інформаційної системи з урахуванням вимог споживачів інформації / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2015. – Вип. 1(126). – С. 75-79.

12. Коваленко А.С. Обгрунтування набору даних для оцінки технічного стану інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Збірник наукових праць Харківського університету Повітряних Сил. – Харків: ХУПС, 2015. – Вип. 1(42). – С.39-41.

13. Коваленко А.С. Експертна система технічного діагностування інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Системи озброєння і військова техніка. – Х.: ХУПС, 2015. – № 1(41). – С. 106-111.

14. Коваленко А.С. Удосконалення методу технічного обслуговування об’єктів інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов,

О.В. Коваленко, О.П. Доренський // Системи озброєння і військова техніка. – Х.: ХУПС, 2016. – № 2(46). – С. 109-114.

15. Коваленко А.С. Метод визначення оптимального комплексу робіт з відновлення працездатності інтегрованої системи технічної діагностики в умовах ресурсних обмежень / А.С. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2016. – Вип. 3(140). – С. 69-72.

16. Kovalenko A.S. Information model and its element for displaying information on technical condition of objects of integrated information system / A.S. Kovalenko, A.A. Smirnov, A.V. Kovalenko, A.P. Dorensky // International Journal of Computational Engineering Research (IJCER). – India: Delhi, 2016. – Volume 6, Issue 1. – P. 21-27.

17. Кожанова А.С. Система технічної діагностики інтегрованих інформаційних систем – обґрунтування необхідності створення, визначення понятійного апарату та напрямів досліджень / А.С. Кожанова, О.А. Смірнов, М.П. Савченко, Д.М. Ізосімов, В.В. Мороз // Створення та модернізація озброєння і військової техніки в сучасних умовах: Тринадцята наук.-техн. конф., 5-6 вер. 2013 р., м. Феодосія: тези доп. – Феодосія: ДНВЦ, 2013. – С. 187-188.

18. Кожанова А.С. Визначення основних напрямків досліджень щодо створення системи технічної діагностики інтегрованих інформаційних систем / А.С. Кожанова, О.А. Смірнов, А.В. Челпанов // Проблемні питання розвитку озброєння та військової техніки Збройних Сил України: IV наук.-техн. конф., 16-20 груд. 2013 р., м. Київ: зб. тез. – Київ: ЦНДІ ОВТ ЗСУ, 2013. – С. 293.

19. Коваленко А.С. Обґрунтування необхідності створення систем технічної діагностики інтегрованих інформаційних систем / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Інформатика та системні науки : V Всеукр. наук.-практ. конф., 13–15 бер. 2014 р., м. Полтава : зб. тез. – Полтава: ПУЕТ, 2014. – С. 292-294.

20. Коваленко А.С. Задачи распознавания ситуаций в системах организационной стратегии интеграции производства и операций

/ А.С. Коваленко, А.В. Коваленко // Комбінаторні конфігурації та їх застосування: XVI міжнар. наук.-практ. сем., 11-12 квіт. 2014 р., м. Кіровоград: зб. тез. – Кіровоград: КНТУ, 2014. – С. 53-55.

21. Коваленко А.С. Створення систем технічної діагностики для автоматизації процесів керування в інтегрованих інформаційних системах / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Проблеми і перспективи розвитку ІТ-індустрії: VI між нар. наук.-практ. конф., 17-18 квіт. 2014 р., м. Харків: зб. тез. – Харків: ХНЕУ, 2014. – С. 241.

22. Коваленко А.С. Визначення понятійного апарату та напрямів досліджень для синтезу систем технічної діагностики інтегрованих інформаційних систем / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Комп'ютерне моделювання у наукоємних технологіях (КМНТ-2014): наук.-техн. конф. з міжнар. участю, 28 -31 трав. 2014 р., м. Харків: зб. наук. праць. – Харків: ХНУ, 2014. – С. 190-193.

23. Коваленко А.С. Основні складові та функції системи технічної діагностики інтегрованих інформаційних систем / Коваленко А.С. // Інформаційні технології та комп'ютерна інженерія: наук.-практ. конф., 4 груд. 2014 р., м. Кіровоград: зб. тез доп. – Кіровоград: КНТУ, 2014. – С. 236.

24. Коваленко А.С. Розробка структури бази даних інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Проблеми і перспективи розвитку ІТ-індустрії: VII міжнар. наук.-практ. конф., 17-18 квіт. 2015 р., м. Харків: зб. тез. – Харків: ХНЕУ, 2015. – С. 15.

25. Коваленко А.С. Дослідження елементів інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Комбінаторні конфігурації та їх застосування: XVII між нар. наук.-практ. сем., 17-18 квіт. 2015 р., м. Кіровоград: зб. тез – Кіровоград: КНТУ, 2015. – С. 5.

26. Коваленко А.С. Метод автоматизованої перевірки результатів вимірювання параметрів об'єкті в інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Стратегія якості у

промисловості і освіті: XI міжнар. конф., 1 – 5 черв. 2015 р., м. Варна, Болгарія.: зб. матер. – Варна: ТУВ, 2015. – С. 423-426.

27. Коваленко А.С. Обґрунтування необхідності створення розподіленої бази даних для забезпечення захисту рухомих повітряних об`єктів / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Перспективні напрями захисту інформації: І всеукр. наук.-практ. конф., 07 вер. 2015 р., м. Одеса: зб. тез доп. – Одеса: ОНАЗ, 2015. – С. 35-39.

28. Коваленко А.С. Розробка інформаційної моделі автоматизованої оцінки технічного стану інтегральної інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Інформаційні технології та взаємодії (IT & I): II між нар. наук.-практ. конф., 3-5 лист. 2015 р., м. Київ: тези доп. – Київ: КНУ ім. Т. Шевченка, 2015. – С. 41-42.

29. Коваленко А.С. Разработка метода усовершенствования технического обслуживания интегрированной информационной системы / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Информационные и телекоммуникационные технологии: образование, наука, практика: II междунар. научн.-практ. конф., 3-4 дек. 2015 г., г. Алматы, Казахстан: сб. труд. – Алматы: КазНИТУ им. К.И. Сатпаева, 2015. – Т.2. – С. 423-427.

30. Королюк Н.А. Оценка временных интервалов работы лица, принимающего решение, на автоматизированном командном пункте / Н.А. Королюк, А.И. Тимочко // Системи обробки інформації. – Х.: ХУПС, 2005. – Вип. 8 (48). – С. 51-54.

31. Костерев В.В. Надёжность технических систем и управление риском: учебн. пособ. / В.В. Костерев. – М.: МИФИ, 2008. – 280 с.

32. Костюков А.В. Підвищення операційної ефективності підприємств на основі моніторингу в реальному часі. / А.В. Костюков, В.М. Костюков. – М.: Машинобудування, 2009. – 192 с.

33. Лазарев А.А. Выбор показателя затрат для анализа сравнительной экономической эффективности техники конечного потребления / А.А. Лазарев,

М.В. Бейлин // Сборник научных трудов ХГПУ.– Х.: ХГПУ, 1999. – Вып. 74. – С. 27-29.

34. Ланецкий Б.Н. Основы теории надежности, технического обслуживания и ремонта вооружения и военной техники: Справочные материалы, часть 1. / Б.Н. Ланецкий, А.А. Посудевский. – Харьков: ХВУ, 1993. – 308 с.

35. Ланецкий Б.Н. Основы теории надежности, технического обслуживания и ремонта вооружения и военной техники: Справочные материалы, часть 2. / Б.Н. Ланецкий, А.А. Посудевский. – Харьков: ХВУ, 1993. – 208 с.

36. Лапсарь А.П. Метод оценки состояния сложных технических объектов для синтеза быстродействующих прогнозирующих систем / А.П. Лапсарь // Измерительная техника. – 2004. – № 2. – С. 7-10.

37. Линейные задачи оптимизации: Учеб. пособие / С.В. Лутманов. – Пермь: ЛИТЕР-А, 2004. – Ч.1. – Линейное программирование. – 128 с.

38. Литвак Б.Г. Экспертные технологии в управлении. Учебное пособие / Б.Г. Литвак. – М.: Дело, 2014. – 318 с.

39. Локазюк В.М. Надійність, контроль, діагностика і модернізація ПК: Посібн. / В.М. Локазюк, Ю.Г. Савченко. – К.: Видавничий центр «Академія», 2004. – 376 с.

40. Лопатников Л. И. Экономико-математический словарь: Словарь современной экономической науки / Л.И. Лопатников. – М.: Дело, 2003. – 520 с.

41. Манухина С.Ю. Инженерная психология и эргономика: хрестоматия / С.Ю Манухина. – М.: Изд. центр ЕАОИ, 2009. –224 с.

42. Мартыненко М.В. Человекомашинные процедуры поддержки организационно–управленческих решений: учеб. пособие СПбГЭТУ / М.В. Мартыненко, О.И. Шеховцов. – СПб, 2012. – 250 с.

43. Мунипов О.В. Эргономика: человекоориентированное проектирование техники, программных средств и среды: Учебник / О.В. Мунипов, В.П. Зинченко. – М.: Логос, 2001. – 356 с.

44. Надеев А.И. Математическая модель эксплуатационной надежности интеллектуальных датчиков / А.И. Надеев, Р.А. Юсупов, Ю.К. Свечников, Д.Р. Юсупов // Измерительная техника. – М: Стандартинформ, 2004. – № 1. – С. 8-11.

45. Надійність техніки. Аналіз надійності. Основні положення: ДСТУ 2861-94 – [Чинний від 1997–01–01]. – Київ: Держстандарт України, 1995. – 33 с. – (Національний стандарт України).

46. Надійність техніки. Терміни та визначення: ДСТУ 2860-94 – [Чинний від 1996–01–01]. – Київ: Держстандарт України, 1994. – 36 с. – (Національний стандарт України).

47. Нейлор К. Как построить свою экспертную систему / К. Нейлор. – М.: Энегроатомиздат, 2007. – 242 с.

48. Про охорону праці: Закон України від 14.10.1992 р. № 2694-XII. Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12#Text

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

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

Додаток А
(обов'язковий)

**Технічне завдання**

**Зміст**





| | | | | | ВКРМ-123.21.0028.00.00.ТЗ | | |
|---|---|---|---|---|---|---|---|
| Вим. | Арк. | № документа | Підпис | Дата | | | |
| Розробив | | Шаповалов А.О. | | | Дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах | Літ. | Аркуш 1 / Аркушів 6 |
| Перевірив | | Доренський О.П. | | | | М | |
| Н. Контр. | | Гермак В.С. | | | | ЦНТУ КІ-20М-1,4 | |
| Затв. | | Смірнов О.А. | | | | | |

## 1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи захисту та відновлення даних в хмарних сервісах.

## 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. № 42-13 від 02.08.2021 року).

## 3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи захисту та відновлення даних в хмарних сервісах.

## 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 захисту та відновлення даних в хмарних сервісах;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

### 5.3 Вимоги до функціональних характеристик

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

### 5.4 Вимоги до архітектури

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

### 5.5 Вимоги до надійності

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

### 5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

### 5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows XP/Vista/7/8/10 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

### 5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows XP/Vista/7/8/10.

#### 5.8.1 Обладнання

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

#### 5.8.2 Мова програмування

Середовище Visual C#.

### 5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

### 5.8.4 Вихідні дані

Робоча програма.

## 6 Вимоги до програмної документації

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

## 7 Економічні вимоги

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

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2021 року.

## 8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинні бути розглянуті шкідливі і небезпечні фактори при роботі з комп’ютером.

## 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 – 128 аркушів.

## 10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

## 11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2021 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 20.12.2021 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

_____ Доренський О.П.

***Дослідження та програмна реалізація***

***системи захисту та відновлення даних в хмарних сервісах***

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 72

Літера: РП

Кропивницький – 2021 року

**Файл RSRaidBase.cs – робота з raid масивами для захисту та відновлення даних в хмарних сервісах**

```
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 i-ой рядка
            int rowNum1this_n = rowNum1 * this.n;
            int rowNum2this_n = rowNum2 * this.n;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

                    return false;
                }

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

                int k_n = k * this.n;

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

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

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

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

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

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

```
                            (colMult != 0)
                        )
                    {
                        for (int i = k; i < (this.n + this.m); i++)
                        {
                            int i_n  = i * this.n;
                            int dIdx = i_n + j;

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

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

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

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

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

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

                    return false;
                }
            }

            return true;
        }

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

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

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

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

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

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

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

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

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

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

                for (int j = 0; j < this.n; j++)
                {
                    //...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 КВАЛІФІКАЦІЙНА РОБОТА",
            "",
            "На тему:",
            "",
            " Дослідження та програмна реалізація системи захисту та відновлення
даних в хмарних сервісах ",
            "",
            "",
            "Керівник: Доренський О.П.",
            "",
            "Розробив: студент Шаповалов Артем Олегович",
            "                  гр. КІ-20М 1,4",
            "",
            "м. Кропивницький 2021"});
            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;

    }
}
```