МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-05

Дипломний проект

здобувача освіти денної форми навчання
РП.05.19.000.ДП

ПІНЬКОВСЬКОГО
АРТЕМА ВІТАЛІЙОВИЧА

м. Одеса
2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**
Освітня програма: **«Розробка програмного забезпечення»**
Група: **4РП-05**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Розробка алгоритмічного та програмного забезпечення для адміністрування надлишкового дискового масиву

Проектний матеріал складається з пояснювальної записки на 74 сторінках та графічного (презентаційного) матеріалу на 25 аркушах (слайдах).

Дипломник _____ (Піньковський А.В.)
Керівник _____ (Кривченко Ю.В.)

Консультанти:
з економічної частини _____ (Копайгородська Т.Г.)
з охорони праці_____ (Чорновол Н.І.)
з дотримання вимог ЄСКД _____ (Петрашова В.І.)
старший консультант _____ (Скорнякова О.В.)

До захисту допущений
Голова циклової комісії _____ (Скорнякова О.В.)
Завідувач відділення _____ (Суліма Ю.Ю.)

Захист «____»_____ 2022 р. Протокол ДКК №_____
Оцінка ДКК _____

Секретар ДКК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення комп'ютерних систем Комісія КТ і ПІ

Спеціальність 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР_____

Беркань І.В.

"_____"_____2022 р.

З А В Д А Н Н Я

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти Піньковському Артему Віталійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка алгоритмічного та програмного забезпечення для адміністрування надлишкового дискового масиву

затверджена наказом по коледжу від "____"_____202___р. №_____

2. Термін здачі закінченого проекту (роботи) _____

3. Вихідні данні до проекту (роботи) _____

1. Специфікації RAID-масивів різних типів;
2. Забезпечити стеження за роботою та налаштуванням RAID-масиву віддалено;
3. Забезпечити сповіщення адміністратора про виявлення помилок у роботі RAID-масиву;
4. Реалізувати програмний модуль "Агент", що зв'язує призначений для користувача інтерфейс з RAID-контролером; 5. Запровадити клієнт-серверну модель взаємодії "Агенту" та "Менеджеру"

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

1. Аналітична частина
2. Розробка структури системи керування надлишковим дисковим масивом
3. Розробка ПЗ для адміністрування надлишкового дискового масиву
4. Опис і експлуатація ПЗ для адміністрування надлишкового дискового масиву
5. Економічна частина 6. Охорона праці; Охорона праці

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

1. Структурна схема клієнт-серверної моделі адміністрування надлишкового дискового масиву
2. БСА роботи модулів "Менеджер" та "Агент"
3. БСА роботи модулів пошуку RAID-контролерів та моніторингу надлишкового дискового масиву
4. Приклади роботи програмного забезпечення для адміністрування надлишкового дискового масиву

6. Консультанти по проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Кривченко Ю.В.		
5. Економічна	Копайгородська Т.Г.		
частина			
6. Охорона праці	Чорновол Н.І.		
Нормоконтроль	Петрашова В.І.		

7. Дата видачі завдання _____

Керівник Кривченко Ю.В. _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1.	Вступ. Постановка задачі проектування	5.05.2022	
2.	Огляд програмних продуктів для керування RAID	7.05.2022	
3.	Аналіз вимог до системи	9.05.2022	
4.	Вивчення моделей представлення даних	11.05.2022	
5.	Розробка алгоритмів модуля Менеджер	13.05.2022	
6.	Розробка алгоритмів модуля Агент	16.05.2022	
7.	Розробка алгоритмів пошуку RAID-контролерів та	18.05.2022	
	моніторингу RAID-масиву		
8.	Вибір і обґрунтування програмних засобів розробки	23.05.2022	
9.	Складання схеми системи керування RAID	25.05.2022	
10.	Розробка схеми проекту та інтерфейсу ПЗ	27.05.2022	
11.	Написання програмного коду та створення інтерфейсу	30.05.2022	
12.	Опис і експлуатація розробленого ПЗ	3.06.2022	
13.	Економічні розрахунки проекту	6.06.2022	
14.	Опис заходів охорони праці та техніки безпеки	8.06.2022	
15.	Оформлення графічної частини та текстової частини	10.06.2022	

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 05. 19 000. 00 ДП	Дипломний проект		
А4			РП 05. 19 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 05. 19 000. 00 ДП		
Розробив		Пінковський А.В.			Розробка алгоритмічного та програмного забезпечення для адміністрування надлишкового дискового масиву	Літ.	Аркуш 4 / Аркушів 74
Перевірив		Кривченко Ю.В.				У Д П	
Н. Контр.		Петрашова В.І.				ОТФК ОНТУ, гр.4РП-05	
Затверд.		Скорнякова О.В.					

ЗМІСТ

ВСТУП

Стрімке зростання продуктивності сучасних процесорів призвело до того, що зараз спостерігається явний дисбаланс між можливостями накопичувачів на жорстких магнітних дисках і потребами процесорів і вирішуваних задач. Якщо не вистачає можливостей одного накопичувача (ємності, швидкодії), частково вирішити дану проблему можливо використовуючи декілька дисків. Звичайно, сама по собі наявність двох або більше накопичувача у настільному комп'ютері (сервері) не є панацеєю: потрібно ще створити на них логічний диск більшої ємності зі збільшеною швидкістю доступу. Крім того, ідеї завадостійкого кодування підводили до думки, що, використовуючи декілька накопичувачів, можна спробувати підвищити надійність зберігання даних, щоб вихід з ладу одного з дисків не призводив до втрати даних.

Жорсткі диски (Hard Disk Drive), які є на сьогоднішній день основними накопичувачами інформації, на жаль не так надійні, як хотілося б. Достатньо гостро стоїть проблема убезпечити свої файли, щоб не довелося вдаватися до відновлення даних.

У результаті вирішення проблеми, пов'язаної з порушенням функціонування обчислювального комплексу і в результаті втратою даних, що знаходяться в пам'яті (вирішення проблеми відмовостійкості), виникла технологія RAID. У 1987 році у статті «A Case for Redundant Arrays of Inexpensive Discs, RAID» (Надмірний масив недорогих дисків) американські дослідники Паттерсон, Гібсон і Катц з Каліфорнійського університету Берклі описали, яким чином можна об'єднати кілька НЖМД в один пристрій так, щоб внаслідок підвищувалися ємність і швидкодія, а відмова окремих дисків не призводила до відмови всього пристрою.

Технологія RAID в даний час набула великого поширення. Якщо зовсім недавно RAID-масиви використовувалися у дорогих серверах масштабу підприємства з застосуванням SCSI-дисків, то сьогодні вони стали стандартом де-факто і для ПК, оскільки всі сучасні материнські плати мають RAID-

контролери. Ознакою того, що використання RAID стає пересічною справою є те, що компанія Western Digital випускає вінчестери підвищеної надійності (напрацювання на відмову складає ~ 1 млн. годин) серії RE (RAID Edition). Відмовостійкість досягається за рахунок надмірності, тобто частина ємності дискового масиву відводиться для службових цілей. Масивом називають кілька накопичувачів, які централізовано налаштовуються, форматуються і керуються. Підвищення продуктивності дискової підсистеми забезпечується одночасною роботою декількох накопичувачів.

Спільну роботу накопичувачів у масиві можна організувати з використанням або паралельного, або незалежного доступу. Таким чином, технології RAID використовуються для захисту від відмов окремих дисків, які (окрім RAID-0) застосовують дублювання даних, що зберігаються на дисках. Для захисту від логічного руйнування даних (руйнування цілісності бази даних або файлової системи), викликаних збоями в устаткуванні, помилками в програмному забезпеченні або невірними діями обслуговуючого персоналу, застосовується резервне копіювання, яке теж є дублюванням даних. Система резервного копіювання призначена для створення резервних копій і відновлення даних.

Система резервного копіювання дозволяє захистити дані від руйнування не лише в разі збоїв або виходу з ладу апаратури, але і в результаті помилок програмних засобів і користувачів. Виконання резервного копіювання є одним з необхідних методів забезпечення безперервності бізнесу. Створення централізованої системи резервного копіювання дозволяє скоротити сукупну вартість володіння IT-інфраструктурою за рахунок оптимального використання пристроїв резервного копіювання і скорочення витрат на адміністрування (в порівнянні з децентралізованою системою). RAID (redundant array of independent disks – надлишковий масив незалежних дисків) – це масив з декількох накопичувачів, керованих контролером взаємозв'язаних швидкісними каналами і сприйманих зовнішньою системою як єдине ціле. Залежно від типу використовуваного масиву може забезпечувати різні ступені відмовостійкості і

швидкодії, служить для підвищення надійності збереження даних і/або для підвищення швидкості читання/запису інформації (в разі RAID 0). При цьому кілька невеликих дисків працюють в системі набагато швидше, ніж кожен з них окремо взятий. Додатково до всього така система з точки зору комп'ютера виглядає як один єдиний диск. Однак, RAID-масив має бути правильно спланованим та конфігурованим, інакше його надійність та швидкодія може виявитися недостатньою.

Основна причина, по якій інформація з RAID-масиву може виявитися втраченою – поломка одного або декількох дисків RAID-масиву, коли перед безпосередньою збіркою підсистеми збереження даних потрібно виконати ремонт жорсткого диска, що вийшов з ладу. Наступна за частотою проблема – вихід з ладу RAID-контролера, далі йдуть помилки RAID-контролера, коли з RAID-масиву "випадають" диски (диску у RAID-масиві став неактивним, отримав статус degraded) і логічні збої – втрата логічних томів RAID або втрачена конфігурація масиву. Для керування дисками та інформацією на них використовується RAID-контролер. Він регулює весь процес зберігання «гарячої копії». Для того щоб можна було конфігурувати RAID-систему і стежити за її працездатністю дана система повинна мати якусь систему керування пристроєм. Для цього існують програмні засоби, призначені для керування і моніторингу RAID-систем.

Дуже доцільним є використання програмного забезпечення для керування і моніторингу надлишкового дискового масиву, яке дозволяє користувачеві стежити за роботою і налаштовувати RAID-систему віддалено з будь-якого комп'ютера не вимагаючи при цьому спеціальних знань з керування RAID-масивом. Також обов'язково потрібно проводити оповіщення адміністратора при виявленні будь-яких помилок роботи надлишкового дискового масиву.

У даній дипломній роботі розглядається створення окремого програмного модуля, що забезпечує зв'язок між надлишковим дисковим масивом і віддаленим комп'ютером, а також стежить за станом надлишкового дискового масиву і нотифікує адміністратора про несправності.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Постановка задачі проектування

При створенні систем зберігання даних на основі RAID-масивів необхідно оснастити RAID-контролери програмою для зручного керування та моніторингу RAID-системи в цілому. Тобто необхідно створити програмне забезпечення, яке дозволить користувачеві стежити за роботою RAID віддалено з будь-якого комп'ютера і не буде вимагати спеціальних знань з керування RAID.

У даному проекті, згідно технічному завданню, RAID-контролери, для яких призначається дане ПЗ, побудовані на базі контролерів Intel i960, і мають однакові команди керування. Тому цю програму можна буде використовувати для всього ряду RAID-контролерів на базі контролерів Intel i960 та їх більш нових версій.

Виділяється три основних завдання, які повинна була вирішити дана система керування.

По-перше, програма повинна бути зручним засобом для керування всією RAID-системою. У зв'язку з цим необхідно заздалегідь продумати всю схему роботи з боку користувача, особливо того, який не має навичок роботи з RAID-системами. Крім загальної концепції необхідно приділити окрему увагу дизайну програми - інтерфейсу користувача. Щоб привернути увагу користувача можна використовувати яскраву графіку, анімовані картинки, звукові ефекти для нотифікації, а також при помилках у системі. Для швидкого зручного способу налаштовувати RAID-масив, треба створити «Майстер первинного завантаження», який буде покроково повідомляти користувача про його подальші дії.

По-друге, програмне забезпечення повинне надавати відомості про роботу RAID-системи. Користувач повинен наочно бачити, як працює його RAID-масив в режимі реального часу, а також мати доступ до інформації про роботу системи в його відсутності – log-файли. Важливий момент – багато адміністраторів керують серверами віддалено по мережі. Тому необхідно реалізувати можливість

доступу до керування не тільки з комп'ютера, до якого підключений RAID-масив, а й з будь-якого іншого комп'ютера по мережі.

По-третє, система повинна будь-яким можливим способом повідомляти адміністратора при поломці у RAID-системі. Наприклад, коли вийшов з ладу будь-який диск, його потрібно якомога швидше замінити на новий, інакше при наступній поломці інформація може бути втрачена.

По суті, програмне забезпечення RAID-системи є єдиним засобом для налаштування контролера на потрібну роботу і для аналізу подальшої роботи цього контролера.

Програмне забезпечення RAID-системи розробляється з урахуванням того, щоб ним могла користуватися людина без спеціальних навичок в адмініструванні RAID-систем. У зв'язку з цим особливим чином продуманий зовнішній дизайн, інтуїтивно зрозумілий інтерфейс.

Також особливу увагу потрібно приділити написанню документації. Файли допомоги будуть розділені на два розділи – допомога при роботі з програмою і поради по використанню RAID-масиву.

1.2 Огляд програмних продуктів для адміністрування надлишкового дискового масиву

Перед початком розробки було необхідно визначити концепцію системи. Для цього потрібно провести дві процедури – оцінити роботу вже існуючих подібних програм і простежити за роботою користувача з установки RAID-системи. На ринку представлено не дуже багато програм для керування RAID-системами. Для докладного аналізу була обрана єдина програма – GAM (Global Array Manager) компанії LSI Logic Corporation. Інші програми були розглянуті поверхнево – в основному з точки зору опису даного продукту.

1.2.1 Програмне забезпечення GAM (Global Array Manager)

Програмне забезпечення Global Array Manager (GAM) розроблялася для компанії LSI Logic Corporation. Дане ПЗ призначено для адміністрування RAID-масивів, побудованих на базі контролера Mylex DAC960 – AcceleRAID® і

eXtreme RAID®. Програма використовує клієнт-серверну структуру – два програмних модуля. Перший модуль – сервер – встановлюється на комп'ютері з RAID-системою і постійно працює. Інший модуль, званий клієнтом, буде працювати тільки під час будь-яких дій над RAID-системою.

Використовуючи доступ через протокол TCP/IP, адміністратор з віддаленої станції може відстежувати і конфігурувати сервер, не переривaючи його роботу. Нові накопичувачі можуть бути додані в систему за допомогою звичайної технології "drag-n-drop". Накопичувачі, що вийшли з ладу, джерела живлення і контролери можуть бути замінені з використанням GAM. Через клієнтський інтерфейс системний адміністратор може отримати повідомлення про критичні події, що відбуваються на сервері або периферійному пристрої. Повідомлення про будь-які зміни потраплять у вікно повідомлень на екрані терміналу адміністратора, а також можуть бути передані по електронній пошті, на мобільний телефон або факс. GAM підтримує всі основні операційні системи, в тому числі сімейства Windows, Netware, SCO UNIX, UnixWare, AIX. Зовнішній вигляд інтерфейсу програми GAM для ОС Windows показаний на рис. 1.1, 1.2.

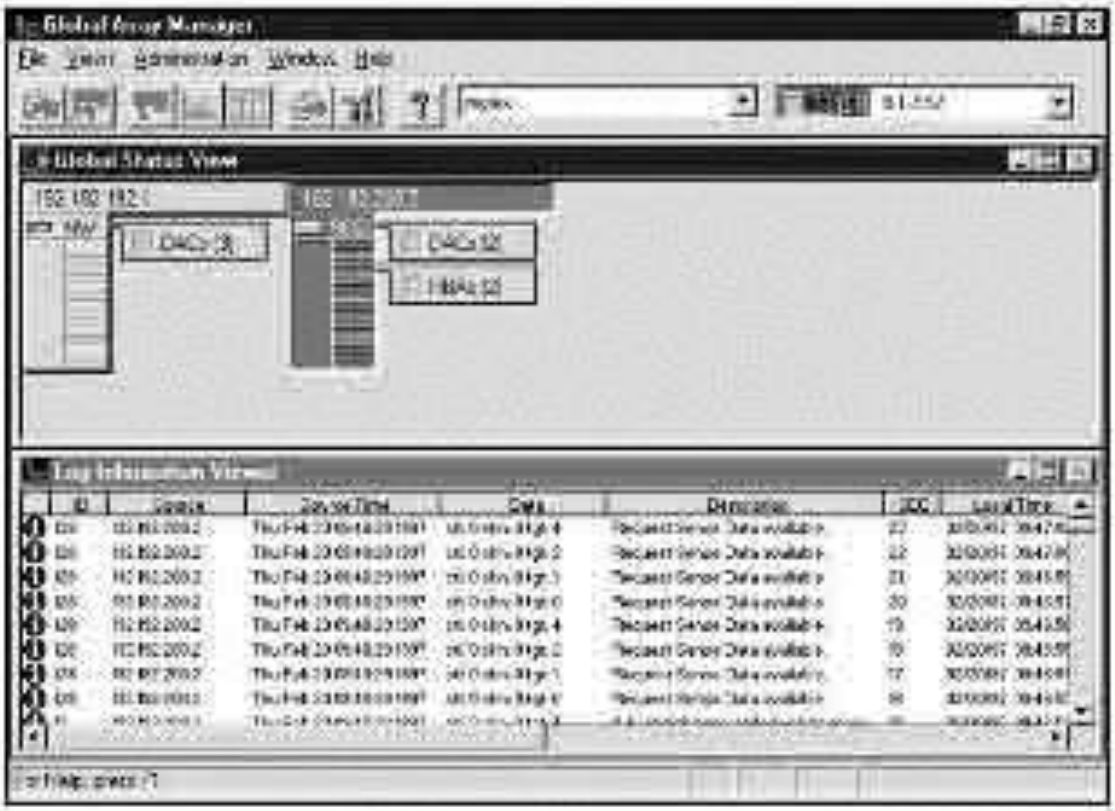

Рисунок 1.1. Зовнішній вигляд інтерфейсу ПЗ Global Array Manager

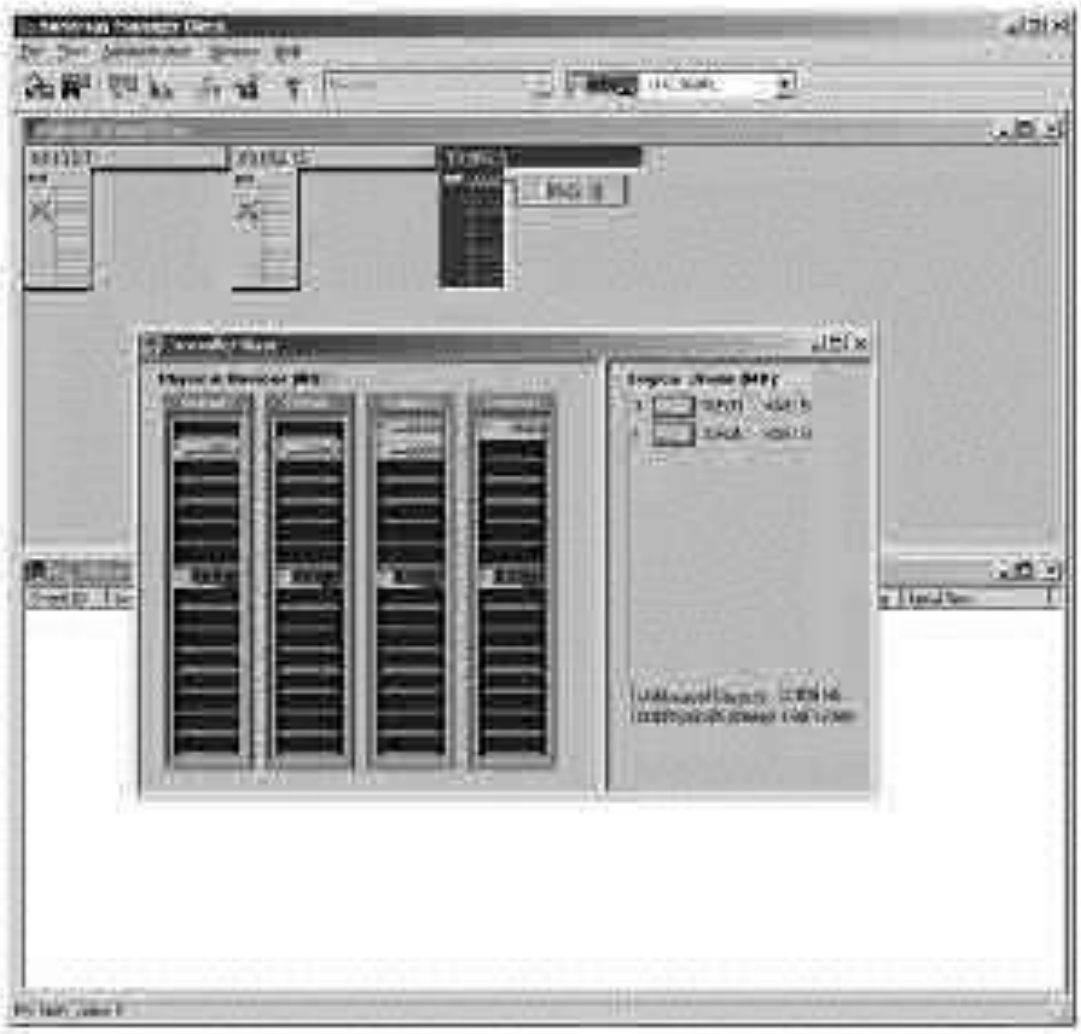

Рисунок 1.2. Режим перегляду стану контролеру ПЗ GAM

Позитивні сторони програми Global Array Manager є такими:

– Віддалене адміністрування. Вся програмна система складається з двох частин – серверна частина, яка встановлюється на комп'ютері з RAID-системою, і клієнтська частина, яка може встановлюватися на будь-якому комп'ютері при з'єднанні по мережі з комп'ютером-RAID. Тим самим адміністратор може віддалено з іншого комп'ютера з'єднуватися з RAID-системою і по мережі передавати команди для керування;

– Зберігання файлів історії. У будь-який момент можна подивитися, як працювала система за останній період часу. Вся інформація про помилки і повідомлення RAID-системи;

– Менеджер конфігурації RAID-системи. Програма пропонує покроково пройти налаштування системи. В цьому випадку мала ймовірність того, що

адміністратор може втратити деякі налаштування. Кожен крок докладно описується документацією, тому буде не складно розібратися навіть початківцю-адміністратору;

– Оповіщення адміністратора про поломку системи. Якщо раптом в RAID-системі сталася поломка, адміністратор відразу ж може отримати сповіщення на e-mail або свій смартфон. Це дуже важлива можливість, тому що при поломці RAID-системи інформація може загубитися і слід негайно вирішити проблему.

Негативні сторони програми Global Array Manager є такими:

– Досить заплутаний і навантажений інтерфейс. Навіть досвідченому користувачеві не відразу зрозуміло, що і де знаходиться в клієнтської оболонці. Система володіє великими функціональними можливостями, але дуже складна в освоєнні;

– Неможливість адаптації даної оболонки під контролери на базі Intel i960 та їх похідних. В основному це неможливо, адже розробка належить конкуруючої компанії.

За оглядом даної програми можна зробити висновок, що найкращим варіантом архітектури додатку є клієнт-сервер. Сервер повинен буде запускатися на комп'ютері з RAID-системою і постійно бути включений, а клієнт може тоді запускатися з будь-якого комп'ютера і з'єднуватися з сервером по TCP/IP протоколу.

Основним мінусом розглянутого ПЗ є незручний інтерфейс, і, внаслідок цього, складність в освоєнні програми. Тому, на етапі проектування необхідно заздалегідь продумати концепцію системи і дизайну програми.

1.2.2 Утиліта LSI Storage Authority

LSA (LSI Storage Authority) має графічний інтерфейс керування напряму з браузера, що робить її кроссплатформним додатком, який прибирає прив'язку до операційної системи. Утиліта працює на HTML5, що робить її дуже зручною.

На рис.1.3 показана інформація по віртуальним і фізичним дискам в утиліті LSA. LSI Storage Authority показує більше інформації по властивостям фізичних

дисків: Status, Exposed As, Product ID, Vendor ID, Serial Number, Shield Counter, Device ID, Usable Capacity, Raw Capacity, SAS Address 0, Drive Speed, Temperature, Revision Level, Power Status, Drive Security Properties, Protection Information та інші.

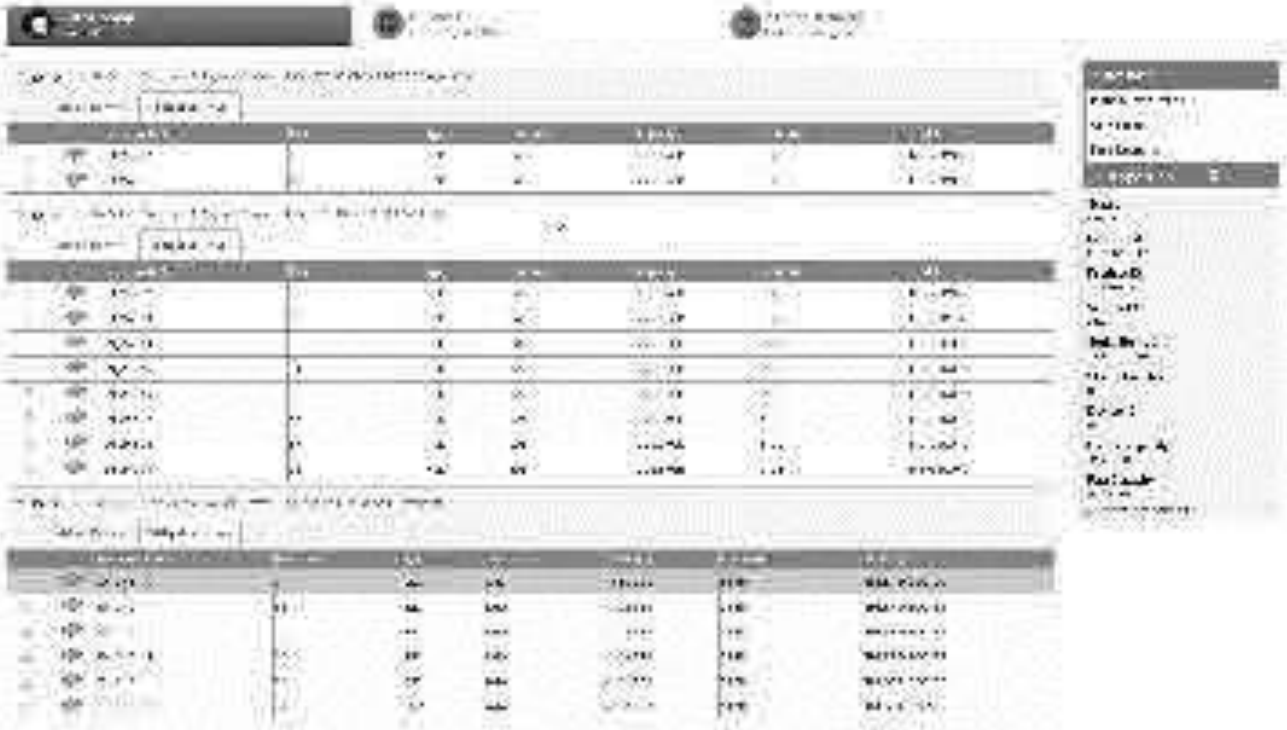

Рисунок 1.3. Інформація по віртуальним і фізичним дискам в утиліті LSA

1.2.3 Утиліта MegaRAID Storage Manager

MegaRAID Storage Manager дозволяє оцінювати стан контролера (Status) – якщо все добре, то статус повинен бути Optimal.

Тут же є область "Action" (рис. 1.4), що дозволяє завантажити конфігурацію, оновити прошивку RAID-контролера, налаштувати звуковий сигнал попередження. Справа є посилання на документацію. Знизу можна бачити поточні події на контролері LSI/Avago. Вкладка "Physical" покаже фізичні диски, вибравши будь-який з них можна подивитися додаткові властивості. Вкладка "Logical" дозволяє взаємодіяти з логічними сутностями. Наприклад можна додати HDD/SSD в існуючий RAID-масив. Можна легко змінити властивості будь-якого Virtual Drive, змінити вид кешування, включити або відключити ініціалізацію.

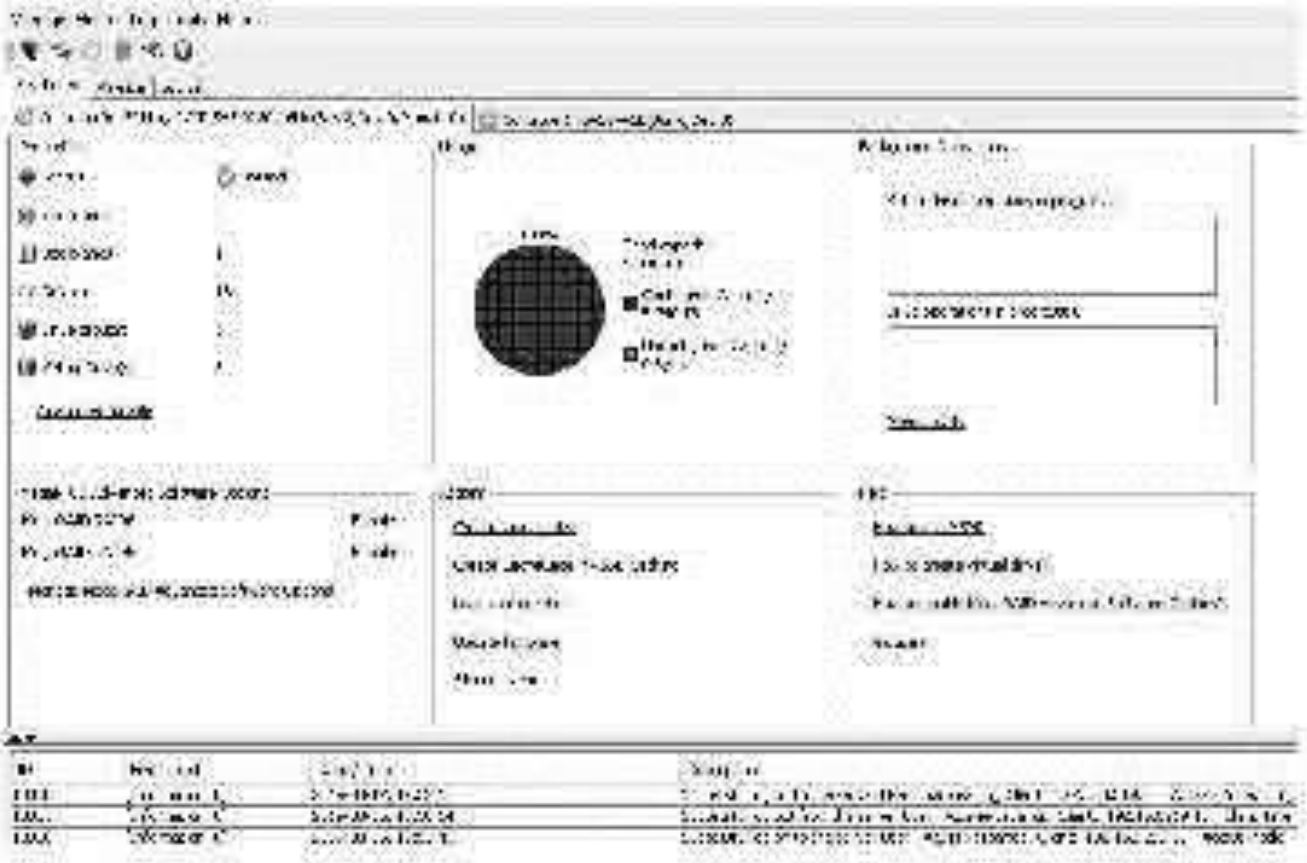

Рисунок 1.4. Розділ Dashboard в утиліті MegaRAID Storage Manager

1.2.4 Утиліта StorCLI

Інструмент StorCLI – це утиліта командного рядка, яка має більший функціонал на відміну від графічних аналогів LSA і MSM. Утиліта дуже проста і зручна. Командний рядок потрібно запускати в режимі адміністратору, можна подивитись зведену інформацію про контролер (рис. 1.5).

StorCLI підтримує операційні системи: Windows Client 10, Windows 8, Windows Server 2016, Windows Server 2012 Red Hat Enterprise Linux, SUSE Linux Enterprise Server, VMware ESXi, Citrix XenServer, CentOS. Утиліта Storecli має засоби моніторингу системи, засоби відлагодження контролера, засоби безпеки контролера, команди по роботі з кешем контролера, команди налаштування контролера, команди моніторингу продуктивності, засоби відображення таблиці поганих блоків, команди налаштування з'єднання.

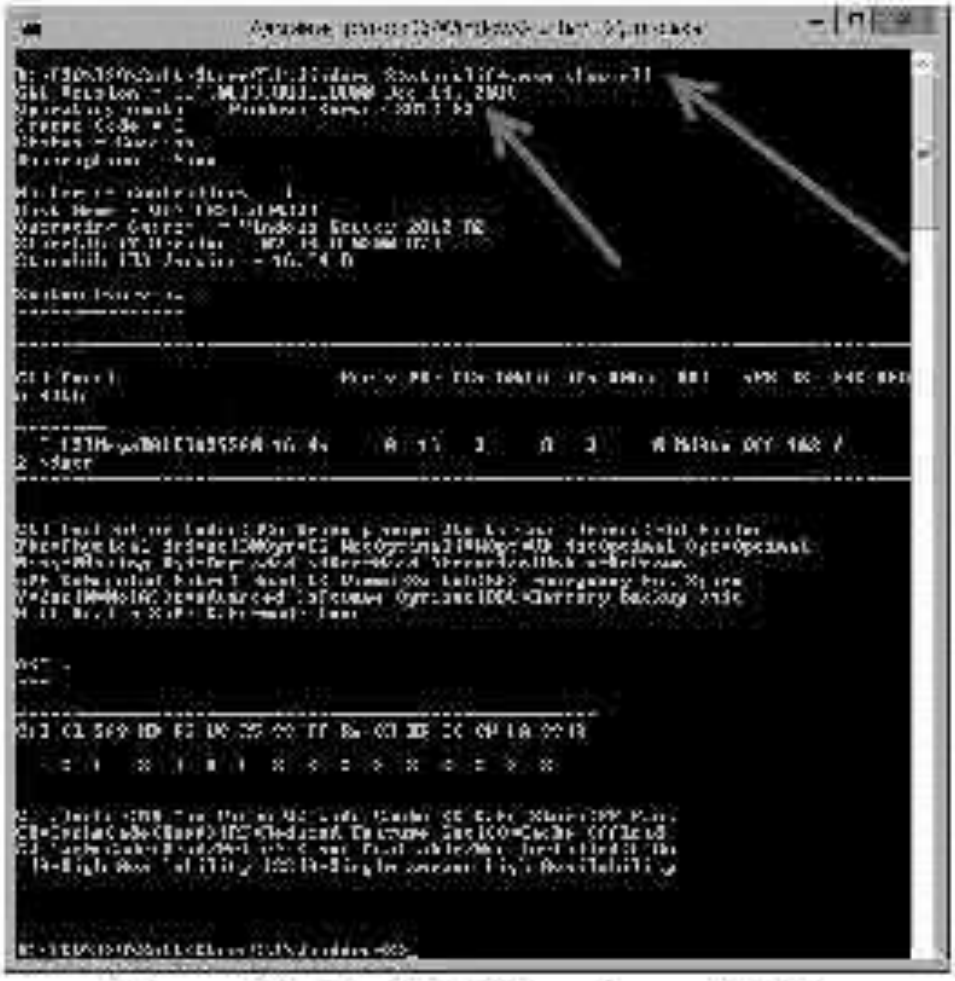

Рисунок 1.5. Розділ Dashboard в утиліті LSA

1.3 Інформаційні потреби користувача

До проектованого програмного забезпечення надлишкового дискового масиву висуваються наступні основні вимоги, які пов'язані із взаємодією ПЗ з користувачем:

1. Встановлення RAID. При підключенні RAID-системи до комп'ютера і після налаштування необхідних драйверів, користувачеві необхідно налаштувати роботу RAID на потрібний режим (RAID 0, RAID 1, RAID 3 і ін.). Після цього готовий для роботи дисковий простір необхідно розбити на потрібні томи. На завершення налаштування необхідно протестувати роботу всієї системи.

2. Керування/Діагностика RAID. Якщо користувачеві необхідно подивитися на стан роботи системи або змінити будь-які параметри, ПЗ має інформативно показати стан роботи системи і надати зручний інтерфейс для зміни налаштувань системи. При цьому як часто буває, адміністратор працює з

комп'ютером, на якому встановлений RAID, віддалено (наприклад, з дому), тому ПЗ має забезпечувати авторизований (захищений) доступ для керування системою по мережі.

3. Обробка помилок. Система повинна негайно повідомити адміністратора виникші про помилки, що виникли в роботі RAID. Сам RAID-контролер не може подати сигнал про несправність, тому ПЗ має забезпечувати безперебійний моніторинг RAID на предмет помилок.

4. Документація ПЗ. Система повинна бути повністю зрозуміла користувачеві. Але незважаючи на це, при виникненні труднощів користувач повинен швидко знайти необхідну документацію як про роботу ПЗ, так і про організацію RAID і режими його роботи.

1.4. Вимоги до ПЗ для адміністрування надлишкового дискового масиву

1.4.1 Складвиконуваних функцій

Створюваний програмний продукт повинен забезпечити виконання наступних функціональних дій:

- початкове налаштування надлишкового дискового масиву;
- щоденний моніторинг стану RAID-системи;
- зміна конфігурації існуючої системи (менеджер дисків, керування дисковим простором, налаштування RAID-контролера);
- можливість віддалено з іншого комп'ютера виконувати керування;
- нотифікація адміністратора про несправності та помилки в роботі надлишкового дискового масиву.

1.4.2 Вимогидо надійності

Зважаючи на те, що система повинна працювати віддалено, повинна бути реалізована система авторизації і захист від несанкціонованого використання системи. Для запобігання перехоплення пароля, переданого через мережу, всі паролі будуть зберігатися в зашифрованому вигляді.

1.4.3 Умови експлуатації і вимоги до складу і параметрів технічних засобів

При віддаленому адмініструванні RAID-системи потрібно запускати два програмних модуля – один на комп'ютері з RAID-системою, інший на комп'ютері адміністратора.

Основною вимогою для використання системи є необхідність постійної роботи програмного модуля, що запускається на комп'ютері з RAID-системою. Якщо цей модуль буде зупинений, то без нього не можна буде виконати з'єднання з RAID-системою і буде неможливим стежити за роботою RAID (відсилати нотифікацію про несправності і вести файли історії роботи RAID).

Для зв'язку обох програмних модулів між собою використовується протокол TCP/IP. Тому для можливості віддалено працювати з RAID-системою, необхідна налаштована мережа для обох комп'ютерів. При адмініструванні RAID-системи з локального комп'ютера підключення до мережі не потрібне.

1.4.4 Вимоги до інформаційної та програмної сумісності

Відповідно до технічного завдання дане програмне забезпечення розробляється під платформу Windows. Програма повинна працювати під основними версіями цієї платформи: Windows 7, Windows 8, Windows 10. Причому серверна частина програми повинна працювати як сервіс (працювати у фоновому режимі).

Потрібно забезпечити можливість подальшого нарощування функцій системи (відкритість для розвитку і методика підключення нових завдань).

Серверна частина програми, яка займається аналізом роботи RAID, повинна бути завжди запущена на комп'ютері з RAID-системою. Якщо цей модуль буде зупинений, то без нього не можна буде зробити з'єднання з RAID-системою і буде неможливим стежити за роботою RAID (відсилати нотифікацію про несправності і вести файли історії роботи RAID).

2 РОЗРОБКА СТРУКТУРИ СИСТЕМИ КЕРУВАННЯ НАДЛИШКОВИМ ДИСКОВИМ МАСИВОМ

2.1 Структурна схема роботи програми

Робота програмного забезпечення системи керування надлишковим дисковим масивом заснована на двох незалежних модулях. Як уже згадувалося, один з них запускається окремо на комп'ютері з RAID-системою, а другий – на комп'ютері адміністратора. Для скорочення будемо називати перший модуль Агентом, а другий – Менеджером.

Менеджер – призначена для користувача сторона програми, яка містить в собі інтерфейс програми, майстер початкової установки, розділ довідки. Менеджер буде керувати RAID-системою за допомогою Агента.

Агент в основному служить для передачі команд від Менеджера RAID-системі і назад. Також Агент буде займатися моніторингом RAID (ведення log-файлу) і нотифікацією адміністратора при помилках.

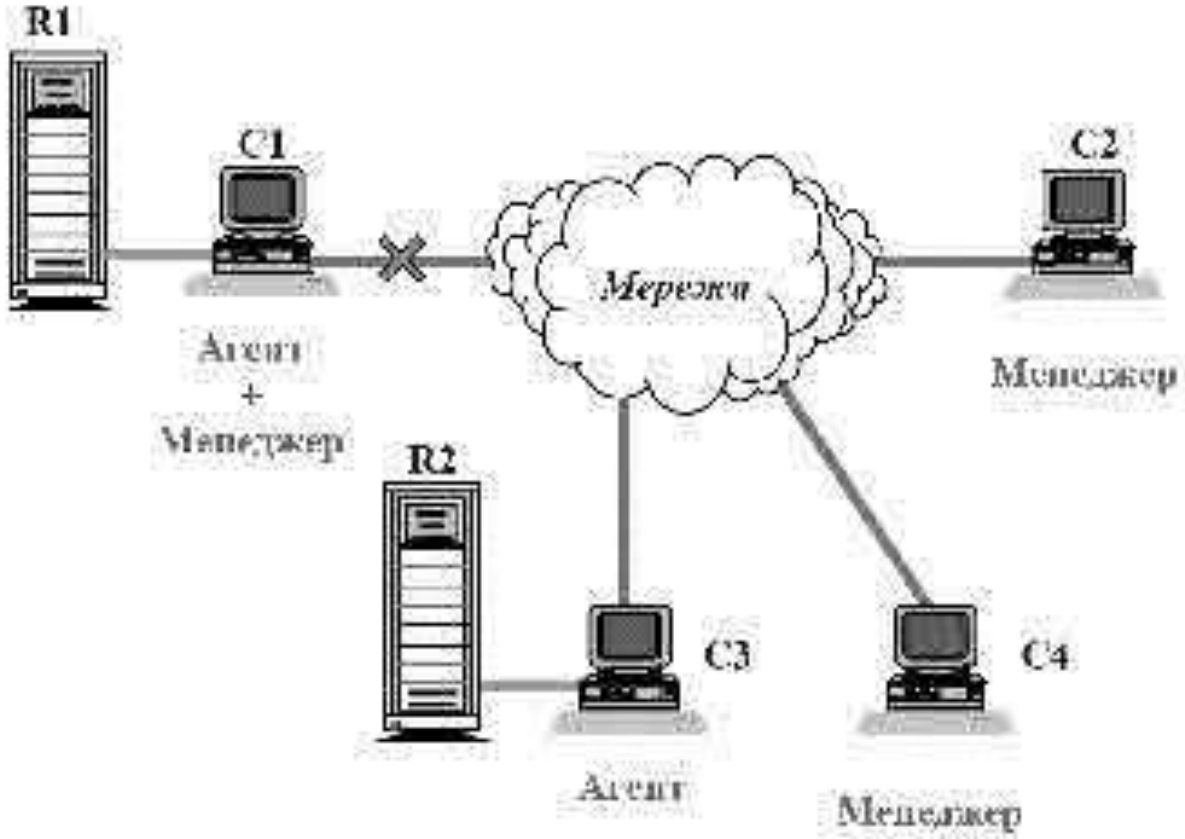

Рисунок 2.1. Структура мережі для роботи ПЗ RAID Manager

Основна структура роботи системи в цілому представлена на рис. 2.1. На ньому показані різні варіанти роботи двох модулів Агент і Менеджер:

- Агент (С3) запущений на комп'ютері і аналізує роботу RAID-масиву R2;
- Менеджер з віддаленого комп'ютера (С2 або С4) може з'єднуватися з мережі з Агентом (С3) для керування роботою RAID-масиву R2;
- Менеджер і Агент запущені на одному комп'ютері С1 для керування роботою RAID-масиву R2. При такому варіанті підключення до мережі не потрібно.

2.2 Структура вхідних і вихідних даних

Основний обмін даними в системі в цілому відбувається по двох каналах:

- між Менеджером і Агентом по мережі по протоколу TCP/IP (команди Менеджера і відповіді Агента);
- між Агентом і RAID-контролером через інтерфейс USB (опитування контролера і відповіді від нього).

Загальна схема обміну даними в проекті проілюстрована на рис. 2.2.

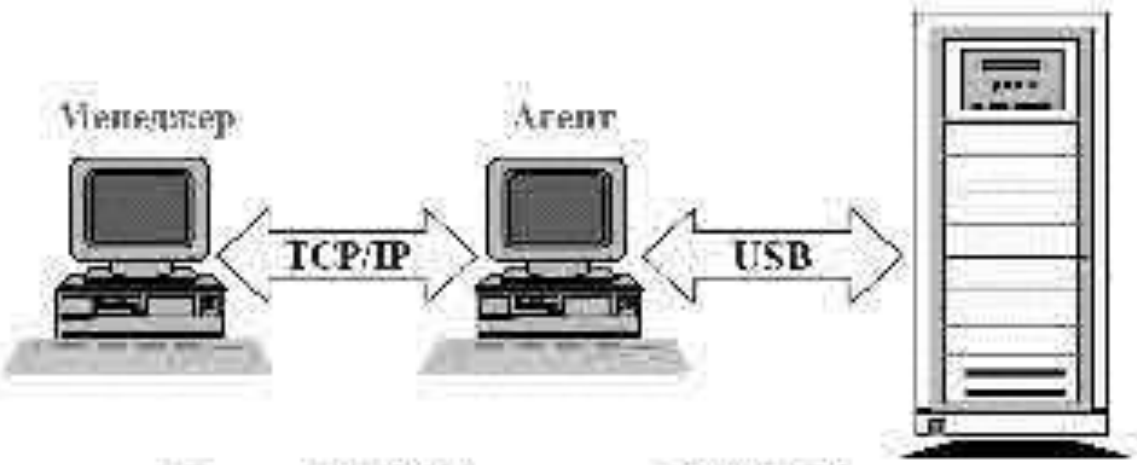

Рисунок 2.2. Обмін даними у ПЗ RAID Manager

Формат даних між Менеджером і Агентом, а також між Агентом і RAID-контролером описаний у підрозділі «Формат даних модуля Агент» даного розділу. Завданням даного дипломного проекту є розробка модуля Агент, тому розглянемо докладніше обмін даних в модулі Агент між Менеджером і RAID-контролером. Розбита на модулі структура модуля Агент показана на рис. 2.3.

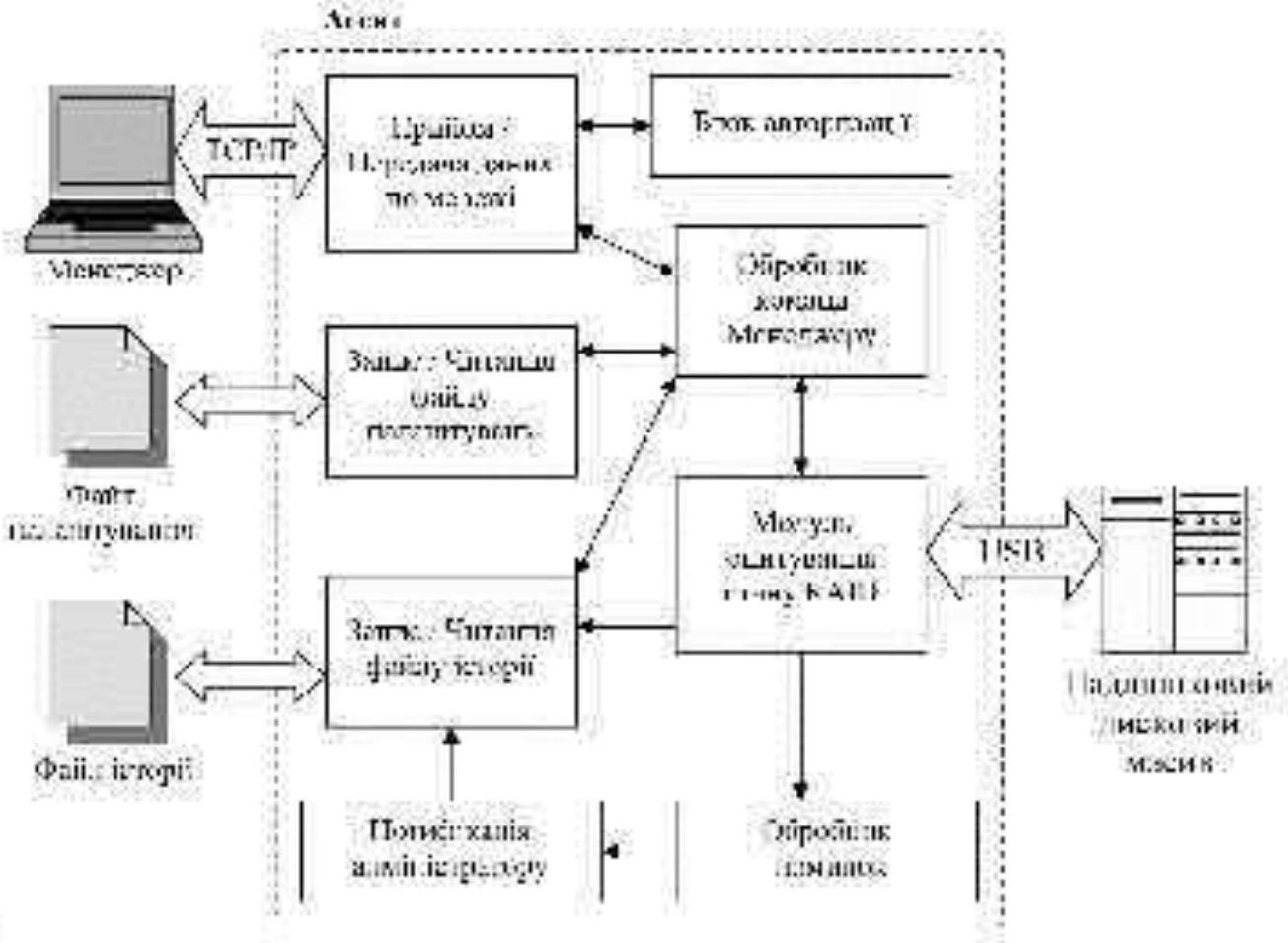

Рисунок 2.3. Обмін даними в модулі Агент

На даній схемі показано, що дані між Менеджером і Агентом проходять через модуль прийому та передачі даних по мережі. Для перевірки підключення Менеджера цей модуль використовує блок авторизації. Всі прийняті дані аналізуються в блоці обробника команд Менеджера. Залежно від типу команди інформація надходить або в блок налаштувань, або в блок файлу історії або в модуль опитування стану надлишкового дискового масиву. Останній служить для того щоб посилати команди RAID-контролера і отримувати від нього відповіді. Якщо при запиті виникає помилка або відповідь контролера містить критичне повідомлення, модуль нотифікації сповістить адміністратора про дану помилку.

3 РОЗРОБКА ПЗ ДЛЯ АДМІНІСТРУВАННЯ НАДЛИШКОВОГО ДИСКОВОГО МАСИВУ

У цьому розділі наводиться опис технологій, використаних в процесі розробки програмного продукту та виконується розробка блок-схем алгоритмів (БСА) програмного забезпечення для віддаленого керування надлишковим дисковим масивом. Для створення ПЗ використовувались принципи об'єктно-орієнтованого програмування. Особливу увагу приділено методам та етапам створення програмних продуктів і методам відлагодження, інструментам створення надійних програмних продуктів.

3.1 Застосування об'єктно-орієнтованого програмування

При написанні програми використовувався об'єктно-орієнтований підхід. Переваги такого методу очевидні:

- об'єктна модель дозволяє в повній мірі використовувати виражальні можливості об'єктних і об'єктно-орієнтованих мов програмування;
- використання об'єктного підходу істотно підвищує рівень уніфікації розробки і придатність для повторного використання не тільки програм, а й проектів, що, врешті-решт, веде до створення середовища розробки. Об'єктно-орієнтовані системи часто виходять більш компактними, ніж їх не об'єктно-орієнтовані еквіваленти. Це означає не тільки зменшення обсягу коду програм, а й здешевлення проекту за рахунок використання попередніх розробок, що дає виграш у вартості і часі;
- можливість розвиватися поступово не призводить до повної переробки програми навіть у разі істотних змін вихідних вимог;
- ризик при розробці складних систем зменшується, оскільки інтеграція закладена ще при проектуванні, а не виконується в кінці розробки.

Основою об'єктно-орієнтованого програмування є об'єктна модель. Вона має 4 головні елементи:

- абстрагування;

– інкапсуляція;

– модульність;

– ієрархія.

Крім головних, є ще 3 додаткових елементи:

– типізація;

– паралелізм;

– збереженість.

Розглянемо ці складові докладніше.

Відповідно до [2], абстракція виділяє істотні характеристики деякого об'єкта, що відрізняють його від інших видів об'єктів і, таким чином, чітко визначає його концептуальні межі з точки зору спостерігача. Абстрагування концентрує увагу на зовнішніх особливостях об'єкту і дозволяє відокремити найістотніші особливості від несуттєвих. Абстракція повинна охоплювати всю поведінку об'єкта, але не більше і не менше, і не приносити побічних ефектів, що лежать поза сферою її застосування.

Інкапсуляція – це процес відділення один від одного елементів об'єкту, що визначають його структуру і поведінку; інкапсуляція служить для того, щоб ізолювати контрольні зобов'язання абстракції від їх реалізації.

Правильне поєднання інкапсуляції з абстрагуванням дозволить, наприклад, легко переписати внутрішню структуру об'єкта, не чіпаючи інтерфейс.

Модульність – це властивість системи, яка була розкладена на внутрішні зв'язки, але слабо пов'язані модулі. Цей принцип доповнює абстрагування і інкапсуляцію, проводячи кордони між абстракціями.

Ієрархія – це впорядкування абстракцій за рівнями. Прикладом ієрархії є спадкування – найважливіший інструмент об'єктно-орієнтованого програмування. Іншим важливим різновидом ієрархії є агрегація, яка визначає відношення "part of".

Поняття типізації взято з теорії абстрактних типів даних. Типізація – це спосіб захиститися від використання об'єктів одного класу замість іншого, або,

прийнятні, управляти таким використанням.

Паралелізм приділяє головну увагу абстрагуванню і синхронізації процесів. Можна сказати, що паралелізм – це властивість, що відрізняє активні об'єкти від пасивних. Майже всі сучасні мови програмування мають засоби для управління нитками і потоками.

Ну і нарешті, останній з елементів об'єктно-орієнтованої моделі – збереженість. Вона представляє собою здатність об'єкта існувати в часі, переживаючи процес, що його породив, і (або) в просторі, переміщаючись зі свого адресного простору.

3.2 Об'єктно-орієнтоване проектування

Розглянемо принципи об'єктно-орієнтованого проектування, яке засноване на об'єктно-орієнтованої моделі. Зазвичай процеси програмування та проектування, відокремлені від процесу управління (або мікропроцес у визначенні Гради Буча), складається з наступних видів діяльності:

- виявлення класів та об'єктів на даному рівні абстракції;
- виявлення семантики цих класів і об'єктів;
- виявлення зв'язків між цими класами і об'єктами;
- специфікація інтерфейсу і реалізація цих класів і об'єктів.

При цьому в мікропроцесі перемішані фази аналізу і проектування. Розглянемо механізм ці процеси більш докладно.

Мета виявлення класів та об'єктів полягає в тому, щоб визначити межі предметної області. Крім того, ця діяльність є першим кроком в продумуванні об'єктно-орієнтованої декомпозиції системи, що розробляється. Цей крок в аналізі застосовується, коли виявляються задачі абстракції, що складають словник предметної області. Такі дії необхідні при проектуванні, коли винаходяться нові абстракції, які є складовими частинами рішення.

Переходячи до програмної реалізації, застосовується процедура виявлення, щоб виявити прості абстракції, з яких будуються більш складні, і виявити загальні риси існуючих абстракцій з метою спростити архітектуру системи.

Результатом виявлення класів та об'єктів є словник даних, що оновлюється по мірі розвитку проекту. Словник даних – центральне сховище абстракцій, що відносяться до системи.

Після виконання цього кроку можна переходити до виявлення семантики класів і об'єктів, тобто визначенню поведінки і атрибутів кожної абстракції. На стадії аналізу цей крок застосовується для розподілу обов'язків між різними видами поведінки системи. На стадії проектування ця процедура дозволяє розподілити обов'язки між частинами реалізації. При реалізації слід просуватися від описів ролей і обов'язків у вільній формі до специфікацій конкретних протоколів для кожної абстракції і, в кінцевому рахунку, – до точних сигнатур (сукупностей ознак) кожної операції. Результатом стає уточнення словника даних. В ході проектування можна виробити специфікації до кожної абстракції, перераховуючи імена операцій в протоколі кожного класу. Наступним кроком стане створення інтерфейсів цих класів на мові реалізації. Прикладом можуть служити заголовки .h в C++. Також складаються діаграми об'єктів і діаграми взаємодій, що передають семантику сценаріїв, які створюються в ході мікропроцесу.

Метою виявлення зв'язків між класами і об'єктами є уточнення межі кожної раніше виявленої у мікропроцесі абстракції і впізнавання всіх сутностей, з якими вона взаємодіє. Ця дія формалізує концептуальне фізичне розмежування між абстракціями, розпочате на попередньому кроці. Цей крок застосовується в аналізі для специфікації зв'язків між класами і об'єктами (включаючи деякі важливі відносини спадкування та агрегації). Існування асоціації має на увазі деяку семантичну залежність між двома абстракціями і можливість переходу від однієї сутності до іншої. Цей етап проектування потрібен, щоб специфікувати взаємодії, які формують механізми архітектури та групування класів в категорії і модулів в підсистеми. Результатами цього кроку є діаграми класів, об'єктів і модулів.

Реалізація класів та об'єктів доводить існуючі абстракції до рівня, достатнього для виявлення нових класів і об'єктів, необхідних на наступному

рівні абстракції. При проектуванні метою реалізації стає створення відчутного уявлення абстракцій шляхом випуску послідовних виконуваних версій системи (макропроцес). Цей крок найчисленно виконується пізніше всіх, адже макропроцес концентрує увагу на поведінці і відкладає рішення про подання. На цьому кроці приймається рішення про подання кожної абстракції і про відображення кожної абстракції у фізичну модель. На початку процесу розробки формулюються тактичні рішення про подання у формі уточнених специфікацій класів. Рішення, що мають загальний інтерес, або відповідні для повторного використання, документуються на діаграмах класів (які показують їх статичну семантику), станів чи взаємодії (що показують їх динамічну семантику). Коли стає ясно, якою мовою реалізовувати проект, починається програмування на псевдокоді або виконуваному коді. Для розкриття зв'язків між логічним і фізичним рівнем, вводяться діаграми модулів, які потім можна використовувати, щоб показати відображення архітектури у програмну реалізацію. З реалізацією пов'язана одна головна дія – вибір структур і алгоритмів, які представляють семантику визначених раніше мікропроцесом абстракцій. На відміну від перших трьох стадій мікропроцесу, зосереджених на зовнішніх уявленнях абстракцій, цей етап акцентує увагу на їх внутрішньому поданні. На стадії аналізу результати цієї дії відносно абстрактні, оскільки розробник не стільки стурбований реалізацією, скільки зацікавлений в знаходженні нових абстракцій, яким можна делегувати обов'язки. На стадії проектування, особливо на пізніх стадіях проектування класів, потрібно переходити до практичних рішень.

3.3 Етапи розробки програмного забезпечення

3.3.1 Постановка задачі

Завдання, яке належить вирішити програмісту, формулюється ним самим або видається йому у вигляді спеціального завдання на розробку програми. Завдання містить формулювання задач, необхідні характеристики розроблюваної програми, вимоги до взаємодії з нею. Видачі такого завдання для великих задач може передувати велика робота науково-дослідного характеру.

Завдання на розробку програми по формі і характеру має бути аналогічним технічному завданню (ТЗ) на розробку якого-небудь технічного продукту. Технічне завдання корисно і в тому випадку, коли замовник і виконавець працюють в одній і тій же кімнаті або навіть є одною особою. Наявність чіткого письмового формулювання буде перешкоджати заміні або відходу в процесі розробки програми від сформульованих в ТЗ вимог на догоду якимось іншим побічним цілям. Крім того, письмово сформульоване завдання робить можливим обговорення, оцінку або узгоджене з замовниками (користувачами) коригування окремих вимог ТЗ в ході розробки програми. ТЗ перешкоджає проникненню в програму таких помилок і прорізів, які можуть бути виявлені тільки після розробки більшої частини програми або вже на стадії аналізу отриманих результатів рахунки. Чим більше формалізованим за характером буде технічне завдання, тим більше шансів, що програма, яка розробляється, буде вирішувати саме ту задачу, яку мав на увазі замовник.

Технічне завдання повинно містити також вимоги чи вказівки, що стосуються принципів перевірки і випробувань готової програми.

3.3.2 Складання проекту

На підставі аналізу технічного завдання програміст вибирає основний метод вирішення задачі, складає загальний проект програми. Обраний підхід до вирішення завдання повинен забезпечувати правильні результати для тих умов функціонування програми, які визначені ТЗ, гарантувати необхідну швидкість роботи, передбачати зручність використання програми і т.п.

У проекті, крім формулювання обраного загального методу розв'язання задачі, характеризуються основні частини проектованої програми, їх функції, взаємозв'язок і послідовність виконання, а також точно визначаються вхідні дані і видаються результати як всієї програми, так і основних її частин. Оскільки кожна програма, що розробляється, як правило, використовується в подальшому не тільки її автором, а й іншими програмістами. складається і проект інструкції для користувачів, в якій фіксується (і, таким чином, може бути заздалегідь

оцінений і виправлений) передбачуваний режим спілкування користувача (і оператора) з програмою.

Для дуже простих задач проектування може здійснюватися програмістом подумки, без фіксації на папері. Навпаки, дуже складні завдання можуть зажадати декількох етапів проектування: передескізне, ескізне, технічне – по аналогії з інженерним проектуванням.

3.3.3 Алгоритмізація

На цей етап іноді дивляться як на допоміжний, підготовчий до виконання наступного етапу, що вважається основним, на якому вирішується написання програми на обраній мові програмування. Введення такого "проміжного" етапу має на меті полегшити написання коду програми і тим самим запобігти виникненню багатьох помилок при його здійсненні для складних задач. Формулювання алгоритму закріплює послідовність основних кроків виконання програми, чітко фіксує функціональний зміст її частин, дозволяє приділити належну увагу простоті логічної структури розроблюваної програми. Етап алгоритмізації є абсолютно необхідним також, в разі, якщо мова, на якій має виконуватись програмування, не цілком освоєна програмістом-розробником і передбачаються труднощі при її використанні на наступному етапі.

При розробці алгоритму необхідно враховувати ресурси використовуваної обчислювальної системи (її швидкість, пам'ять) і можливості застосовуваної для розв'язання задачі операційної системи. Алгоритми для нескладних задач, вимоги яких до ресурсів невеликі, є зазвичай машинно-незалежними.

В ході розробки загального алгоритму використовується деяка спеціальна мова, який за своїм характером є проміжною, перехідною між неформальним, словесним способом викладу методу розв'язання задачі на етапі 1 і формальною алгоритмічною мовою для програмування на етапі 3. Проміжна мова повинна поєднувати в собі, з одного боку, наочність для відображення змісту і сенсу виконуваних в алгоритмі дій (що виконується) і, з іншого боку, формалізм для зазначення конкретних операцій і послідовності їх виконання (як виконується).

У якості такої проміжної мови зазвичай використовують блок-схеми, які дозволяють найбільш наочно уявити логічну структуру програми, що розробляється, взаємозв'язок окремих частин програми, умови або кратність виконання таких частин. Для відображення обчислювальної (арифметичної) сторони програми використовуються звичайні математичні засоби або елементи алгоритмічних мов, а в найзагальніших блок-схемах – просто словесне формулювання; іноді використовуються і всі ці способи разом.

Для досить складних програм алгоритмізація проводиться в кілька кроків з метою поступової деталізації алгоритму. Критерієм закінчення деталізації при цьому є те, що для кожного псевдо-оператора в отриманому на черговому кроці алгоритмі програміст вже має чітке і конкретне уявлення про те, як цей оператор алгоритму може бути виражений засобами вибраної мови програмування на етапі 3. Для простих задач зазвичай розробляють блок-схеми на двох рівнях: загальна блок-схема програми та блок-схеми окремих частин (блоків) програми.

Після останнього кроку деталізації алгоритму (а іноді і після окремих великих кроків) проводиться перевірка отриманого алгоритму для виявлення допущених помилок. Методи контролю алгоритму аналогічні деяким методам контролю програми.

В ході розробки алгоритму, можливо, доведеться уточнювати або змінювати рішення, прийняті на етапі 1, і в цьому випадку такі зміни обов'язково вносяться в проект, який завжди повинен відповідати алгоритму, що розробляється.

3.3.4 Програмування

У разі, коли на попередньому етапі був отриманий детально розроблений алгоритм, складання програми на обраній для програмування мові (алгоритмічній мові високого рівня, автокоді, мові асемблера або машинній мові) зводиться до перекладу цього алгоритму на мову програмування. Основні труднощі і, отже, причини помилок на цьому етапі полягають, по-перше, в необхідності знання всіх вимог і обмежень вибраної мови програмування і, по-

друге, в необхідності постійної уваги до багатьох деталей мови, які доводиться враховувати в ході написання програми. Якщо етап 2 було виконаний неякісно і алгоритм представлений недостатньо детально, то його доробку доведеться виконувати «на ходу», під час програмування. Це ускладнить процес програмування-перекладу і призведе до виникнення додаткових помилок в програмі. Чим більше процес програмування буде походити на переклад, ніж більш механічним буде такий переклад, тим легшим буде складання програми і тим менше виникне помилок на цьому етапі. Етап програмування, тобто власне процес написання програми, останнім часом все частіше називають кодуванням, що підкреслює його механічний, нетворчий характер.

Після складання програми проводиться її перевірка для виявлення та виправлення помилок, внесених на цьому етапі. Якщо при перевірці виявляються помилки, допущені на попередньому етапі (2), то відповідні виправлення вносяться і до алгоритму, оскільки до нього ще доведеться звертатися на наступних етапах, і тексти алгоритму та програми повинні відповідати один одному.

3.3.5 Препарація

Після складання програми проводиться її перенесення на машинні носії, тобто підготовка програми до виконання її на комп'ютері; будемо називати цей етап препарацією. Для попередження і скорочення помилок препарації текст програми повинен бути написаний ясно і чітко: чим більш недбалим буде написаний текст програми, тим більше помилок виникне у препарованій програмі. Перевірка правильності препарації здійснюється роздруківкою програми, введеної у комп'ютер з використаних носіїв, і подальшої звірки з вихідним текстом.

Роздруківка програми може проводитися за спеціальною автономною (поза операційної системи) працюючою на машині програмою або за допомогою однієї із системних сервісних програм. Текст вихідної програми можна отримати і при подальшій трансляції складеної програми, але попередня роздруківка

зручніше, оскільки при виявленні грубої синтаксичної помилки транслятор може припинити свою роботу і друк тексту або видати надзвичайно багато діагностики, яка є дуже складною для розбору і у більшості своїй марною.

Як бачимо, розглянутий етап на відміну від попередніх вже вимагає для свого здійснення безпосереднього використання комп'ютеру або, принаймні, її зовнішніх пристроїв.

3.3.6 Трансляція

Транслятор в ході здійснення трансляції, поряд із друком програми, що транслюється, здійснює пошук синтаксичних помилок в програмі і, в разі їх виявлення, друкує діагностику, що допомагає подальшій локалізації помилок. Трансляція, а разом з нею і пошук синтаксичних помилок, можуть бути припинені, якщо знайдена дуже груба (з точки зору транслятора) помилка. Відсутність синтаксичних помилок не говорить про те, що в програмі немає помилок препарації (наприклад, замість знаку * був + або записана не та буква і т.п.). Тому ретельна звірка надрукованої програми з вихідним текстом завжди необхідна на даному або попередньому етапі.

Перші трансляції знову складеної програми проводяться зазвичай з включенням таких режимів транслятора, які дозволяють отримати текст програми разом з додатковою інформацією про програму (наприклад, з таблицею використовуваних в ній ідентифікаторів) для найбільш повної її звірки.

У тих випадках, коли в розпорядженні програміста є кілька трансляторів, спочатку вибирається той, який представляє більше можливостей для проведення відлагодження, що починається. Після закінчення відлагодження, уже за допомогою оптимізуючого транслятору проводиться перетрансляція з метою виготовлення примірника програми, яка використовується в подальшому для рахунку.

3.3.7 Відлагодження

На етапі відлагодження проводиться виявлення за допомогою комп'ютеру помилок в програмі і їх виправлення. Етап відлагодження можна розділити на три підетапи:

1. Контроль правильності програми;

2. Локалізація помилок;

3. Виправлення помилок.

На першому підетапі шляхом запуску на машині спеціальних контрольних прикладів встановлюється факт відсутності або, в іншому випадку, наявності помилок в програмі. Тут мова йде про змістовні (семантичні) помилки, які не виявляються при трансляції програми.

На другому підетапі точно встановлюється місце, де в програмі допущена помилка (помилки), наслідки якої виявились під час виконання попереднього підетапу.

На останньому підетапі проводиться виправлення помилок, виявлених на другому підетапі. Виправлення вносяться як в програму, так і в алгоритм, якщо він зачіпається цими виправленнями.

Перераховані підетапи можуть повторюватись багаторазово (включаючи і етап трансляції, точніше перетрансляції), до тих пір поки контроль покаже, що помилок в програмі, мабуть, немає.

3.3.8 Оформлення програми

Для можливості експлуатації програми ким-небудь крім автора вона повинна бути оформлена: складений її опис, виготовлені машинні носії для передачі програми користувачам. В опис включається інструкція по використанню програми, викладається застосований метод рішення, наводяться алгоритми (іноді і текст програми), а також контрольні приклади з еталонними результатами. Наявність опису програми дозволяє не тільки успішно експлуатувати її тривалий час, але і проводити її модернізацію і використовувати в подальших розробках. Основну частину опису становлять матеріали, з якими

йшла робота на попередніх етапах розробки (проект розробки та опис методу рішення, загальна блок-схема, алгоритми, проект інструкції для користувача і т. п.). Тому для прискорення етапу оформлення всі перераховані матеріали завжди повинні бути в робочому стані і за змістом цілком відповідати одна одній і програмі, що відлагоджується, крім того, вже на етапах розробки їх потрібно представляти в такому вигляді, щоб вони могли бути використані для опису програми без додаткових переробок.

У разі, коли програма проста і призначена для експлуатації тільки її автором, оформлення програми може проводитися вже після проведення розрахунків по ній, одночасно з виготовленням звіту (див. нижче).

3.3.9 Експлуатація

Після закінчення відлагодження і оформлення програми починається її експлуатація: проводиться розрахунки за її допомогою, зазвичай багаторазові. Перші отримані результати реальних розрахунків піддаються ретельному аналізу, щоб переконатися в придатності використаного методу та встановити узгодженість отриманих результатів з наявними даними і теорією. Якщо правильність одержуваних результатів не викликає сумнівів і ефективність програми задовільна, то її експлуатація продовжується в міру необхідності. Але трапляється і так, що доводиться знову розглядати питання правильності розробленого алгоритму або придатності реалізованого методу, і тоді вся робота може повернутися до початку.

3.3.10 Звіт про роботу

На підставі результатів, отриманих в ході експлуатації програми, складається звіт про виконану роботу, оцінюється обраний спосіб розв'язання задачі та ефективність програми, публікуються наукові висновки.

3.3.11 Модернізація

Якщо розробник програми постійно працює в деякій області науки або техніки, то зазвичай рано чи пізно настає такий момент, коли перед ним постає питання про модернізацію старої програми або про складання нової, розвиваючої ідеї, реалізовані в першій програмі. Модернізація програми проходить ті ж етапи, що і розробка, і починається зі складання технічного завдання на модернізацію. Успішне здійснення модернізації залежить від того, наскільки легко можна буде при розробці нової програми використовувати блоки старої програми і вносити в них зміни. Швидке виконання такого роду робіт залежить, в свою чергу, як від структури модернізованої програми, так і від якості її оформлення (наявність опису програми, докладних алгоритмів, пояснень до програми і т. п.).

Всі перераховані етапи явно присутні і добре проглядаються при розробці досить складних програм. Для простих завдань деякі етапи можуть поєднуватися один з одним або проходити непомітно без будь-якої чіткої фіксації. Наприклад, для елементарних завдань перші два або навіть три етапи часто відсутні або з'єднуються в один підготовчий етап. Цим, зокрема, можна пояснити те, що початківці програмісти, котрі навчалися програмуванню на елементарних задачах, при переході до складання складних програм виявляються не готовими до розробки технічного завдання, проекту, загальних алгоритмів. Крім того, перші етапи можуть бути значно скорочені, якщо програміст отримує завдання на програмування, яке містить вже загальний алгоритм програми, що розробляється. З іншого боку, для складних задач може бути кілька підетапів алгоритмізації, відлагодження (автономне, комплексне) і додається період дослідної експлуатації.

3.4 Загальні схеми алгоритмів роботи програми

3.4.1 Загальна схема алгоритму роботи модуля Менеджер

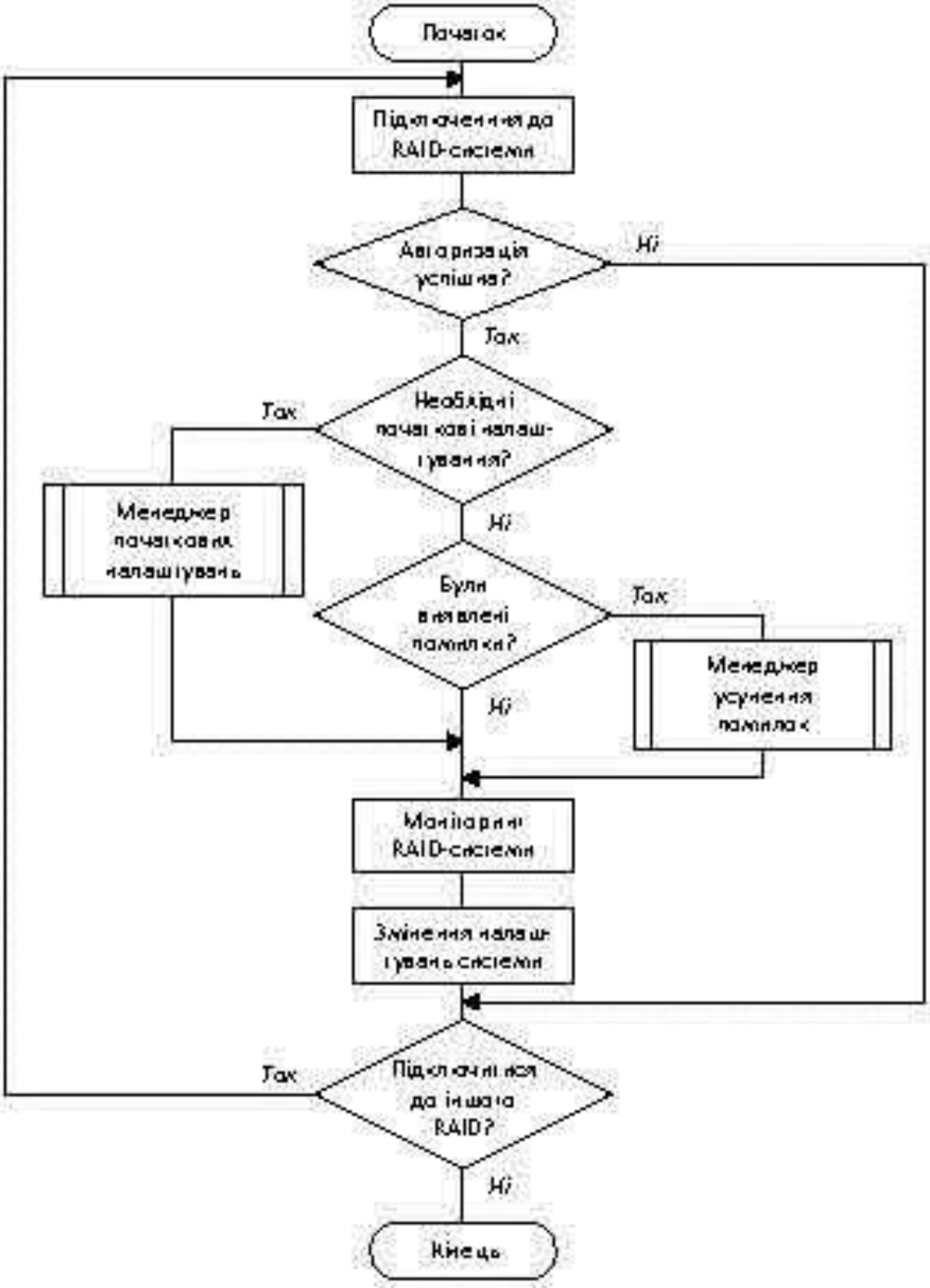

Рисунок 3.1. Загальна схема алгоритму роботи модуля Менеджер

На рис. 3.1 представлена загальна схема алгоритму роботи модуля Менеджер. Крім основного вікна програми передбачені допоміжні інтерфейси для початкових налаштувань програми, а також спеціалізований модуль для усунення помилок. Система сама активізує ці модулі при необхідності. При виході з програми є можливість підключитися до іншого RAID-контролера, якщо адміністратор обслуговує декілька надлишкових дискових масивів.

Загальна схема роботи можна описати наступним чином. При запуску програми модуль Менеджер з'єднується з комп'ютером, на якому встановлено надлишковий дисковий масив і відбувається авторизація користувача. Після успішної авторизації, якщо система не була налаштована до цього, користувачеві пропонується налаштувати систему за допомогою менеджера початкового налаштування. При появі помилок у надлишковому дисковому масиві система викликає діалог з повідомленням про цю помилку і пропонує спосіб щодо усунення цієї проблеми. В іншому система дозволяє здійснювати моніторинг надлишкового дискового масиву і виробляти деякі настройки працюючої системи. Після закінчення роботи з даним RAID-масивом можна під'єднатися до іншого RAID-контролера.

3.4.2 Схема алгоритму роботи модуля Агент

Загальна схема алгоритму роботи модуля Агент представлена на рис. 3.2. Більш докладні схеми алгоритмів для пошуку підключених RAID-контролерів, обробки команд Менеджера і моніторингу надлишкового дискового масиву будуть розглянуті далі. Даний модуль працює як сервер, тому вихід з нього організований тільки з примусу від користувача або якщо не було знайдено підключених RAID-пристроїв до даного комп'ютера. При запуску Агент насамперед перевіряє, чи є вже знайдені RAID-контролери, підключені до цього комп'ютера. Якщо таких пристроїв не виявлено, активізується модуль для пошуку RAID-контролерів. Схема роботи цього модуля буде розглянута нижче. Якщо пошук не дав позитивних результатів, то робота модуля Агент зупиняється.

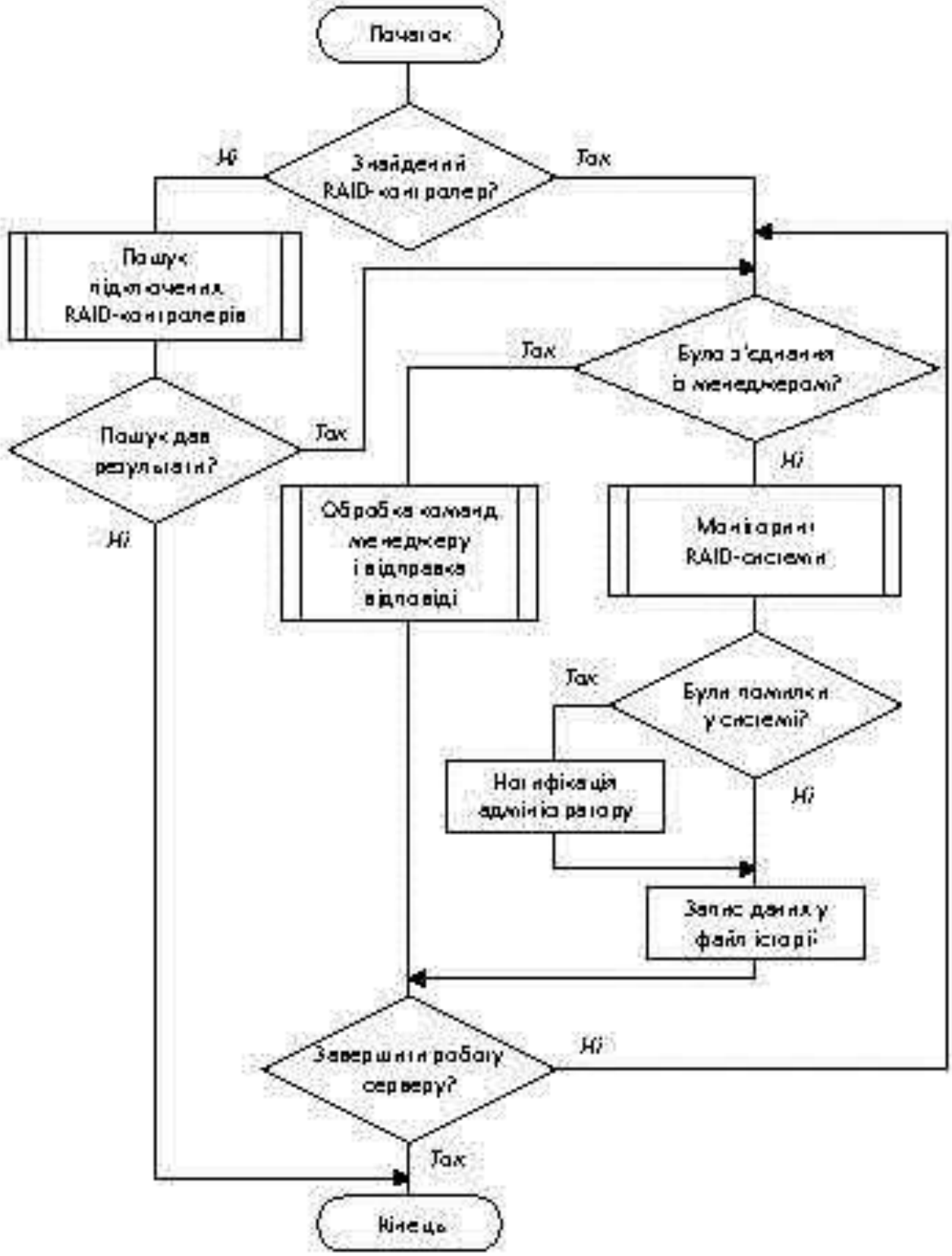

Рисунок 3.2. Схема алгоритму роботи модуля Агент

Далі модуль постійно через вказаний в налаштуваннях проміжок часу опитує RAID-масив і записує ці дані в файл історії. Якщо сталася якась помилка в RAID-системі, то модуль негайно зазначеним в настройках методом посилає

повідомлення адміністратору про цю помилку. Під час отримання запиту на з'єднання від Менеджера модуль встановлює з'єднання і передає дані від Менеджера RAID-контролеру і назад. Більш докладно про алгоритми роботи модуля Агент буде йти мова у наступному підрозділі.

3.5 Схеми алгоритмів модуля Агент

У даному підрозділі докладно описані алгоритми роботи програмного модуля Агент.

3.5.1 Пошук підключених RAID-контролерів

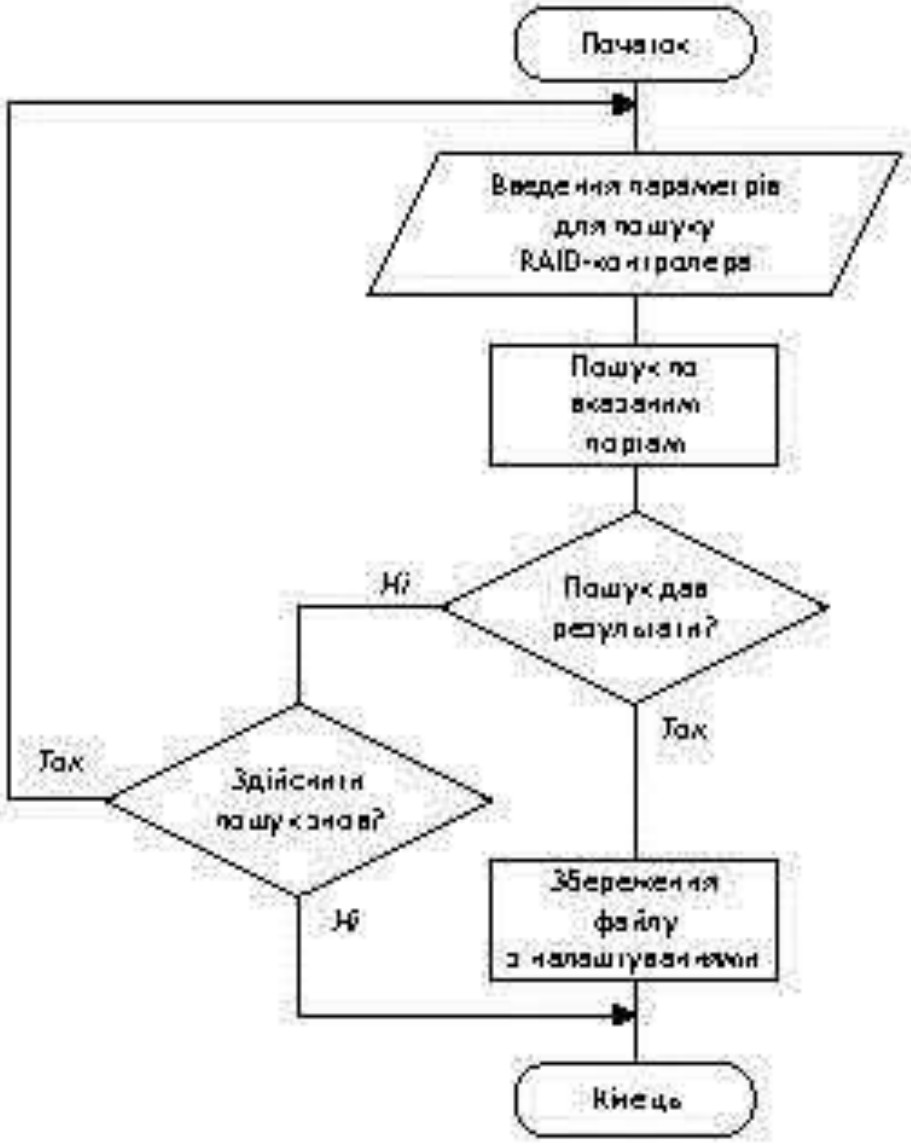

Рисунок 3.3. Схема алгоритму для пошуку підключених RAID-контролерів

Даний модуль запускається, якщо при старті Агента не було знайдено файлу налаштувань, або дані налаштування не вірні. Схема алгоритму цього модуля показана на рис. 3.3. Спочатку необхідно задати параметри для пошуку RAID-контролера. На наступному кроці програма буде сканувати зазначені USB-порти по заданих параметрах.

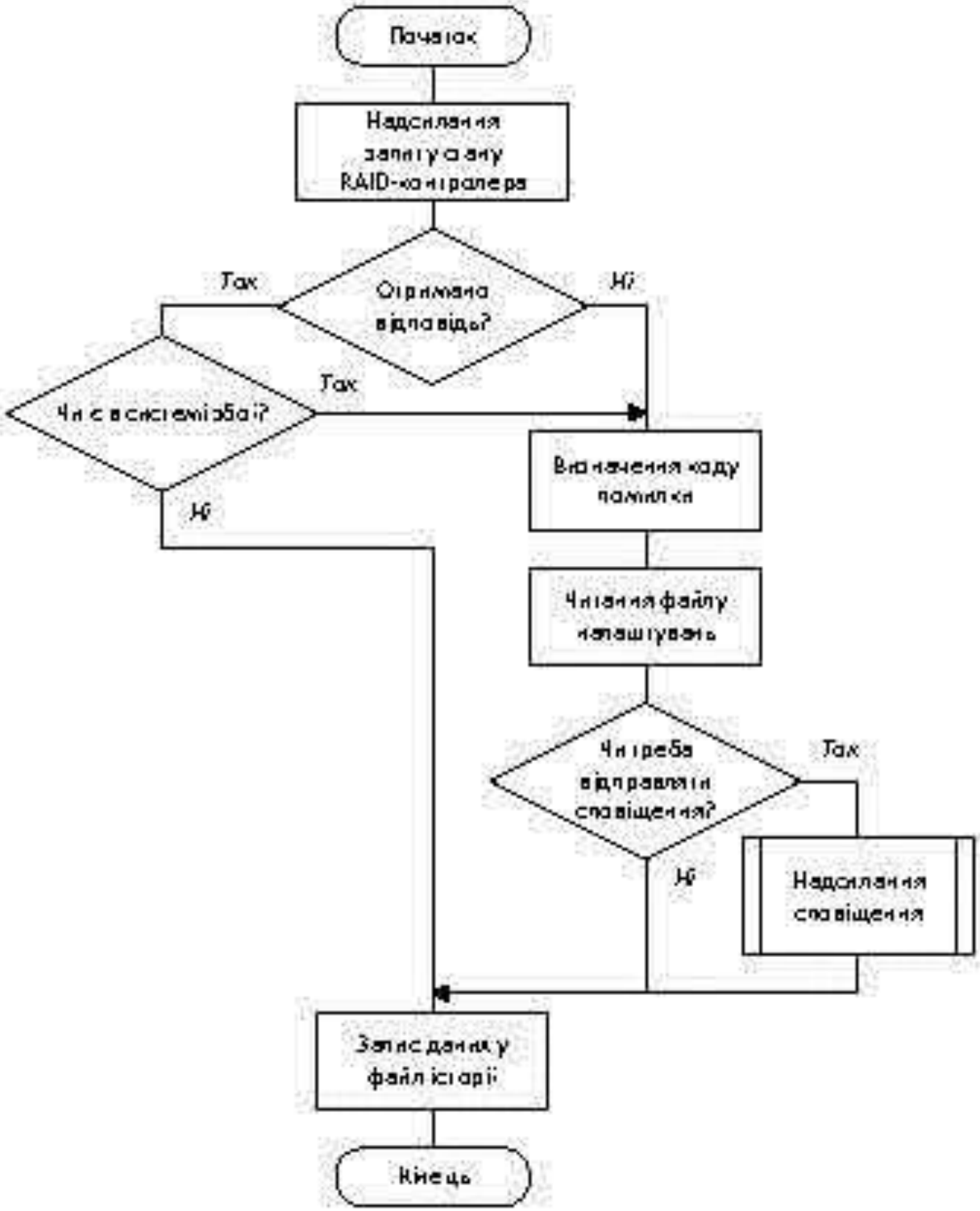

Рисунок 3.4. Схема алгоритму моніторингу надлишкового дискового масиву

Якщо пошук закінчився невдало, система запропонує провести пошук заново, змінивши параметри пошуку. Після того як RAID-контролер був знайдений, система записує параметри в певний файл, який буде використовуватися при запуску Агента.

3.5.2 Моніторинг надлишкового дискового масиву

Моніторинг надлишкового дискового масиву здійснюється за таймером. Проміжок часу для моніторингу задається в налаштуваннях або через інтерфейс Менеджера чи Агента. На рис. 3.4 розглянуто один цикл для опитування надлишкового дискового масиву

Основний момент в цьому циклі займає нотифікація адміністратора, якщо була знайдена помилка. При виявленні помилок в роботі системи визначаємо, чи треба відправляти сповіщення для адміністратора про несправність. Для цього необхідно спочатку визначити код цієї помилки. У файлі налаштувань для сповіщення для кожної категорії помилок вказується тип сповіщення: E-mail, Broadcast або в log-файл. Якщо помилок не було знайдено, у log-файл записується рядок про успішну перевірку роботи системи.

3.6 Формат даних модуля Агент

Розглянемо докладно формат даних модуля Агент, а також формат команд для обміну інформацією між Агентом і Менеджером і між Агентом і RAID-контролером.

3.6.1 Файл конфігурації Агента

Файл конфігурації потрібен для визначення підключеного до комп'ютера RAID-контролера.

```
[Mail]

TYPE = USB
PORT = 1
DATABITS = 8
STOPBITS = 1
PARITY = 0
```

```
LogFileName = 000032599000004.log
LogFileSize = 1024
DelayTime1 = 0
DelayTime2 = 15
DelayTime3 = 30
DelayTime4 = 60
Cnt = 1
skin = 1
email_1=saga@rusoft.ru
emessage_1 = HI
address_1 = SAGA
bmessage_1 = HI
```

3.6.2 Формат файлу історії

У файл історії записуються всі події за форматом:

[День/місяць/рік(2) години:хвилини:секунди]: повідомлення.

Наприклад:

```
[05/05/22 14:39:18]:New connection opened
[05/05/22 14:39:18]:GUI detected via:127.0.0.1
[05/05/22 14:39:18]:GUI connected at:127.0.0.1
[05/05/22 14:40:08]:Login accepted:127.0.0.1: session opened
```

3.6.3 Формат команд RAID-контролера

У таблиці 3.1 наведено фрагмент з опису до контролера ACS-9900 з описом команд, що посилаються RAID-контролеру.

Таблиця 3.1. Формат команд контролера ACS-9900

Command	*Description*
Get page	Get the RAID information page
Edit RAIDi	Edit RAID1~4, If Argument code is No, This Command is Aborted
Set RAIDi Level	Set RAIDi Level---i=1or2or3or4 0—0xi0, 1—0xi1, 3—0xi3, 5—0xi5, 0 + 1—0xi6, None -- else
Select RAIDi Disk Member	Select RAIDi Disk Member (i = 1or2or3or4) if xx = 01 Disk01 is selected, xx = 03 Disk03 is selected.....

У таблиці 3.2 наведено фрагмент з опису до контролера ACS-9900 з варіантами відповідей від RAID-контролера.

Таблиця 3.2. Формат відповідей контролера ACS-9900

Message	*Meaning*
INITIALIZING ARRAY X ...:	Initializing Array x
INIT ARRAY ERROR	Error occurred while initializing Array
INIT ARRAY OK	Array initializing successful
ADD (X): ERROR !:	Error occurred while rebuilding Slot x
CHECK ARRAY ERROR !:	Error occurred while checking Array
CHECK ARRAY OK.	Array checked without any problems
HOST CHAN X: TEST OK	Host channel x is ok
HOST CHAN X: TEST FAIL	Error occurred while testing Host channel x

3.7 Вибір і опис платформи проектування

Відповідно до технічного завдання для розробки програмного забезпечення була обрана платформа Windows.

Програма повинна працювати під основними версіями цієї платформи: Windows 7, Windows 8, Windows 10. Причому серверна частина програми повинна працювати як сервіс (працювати у фоновому режимі).

Надалі (не в рамках поточного проекту) програмне забезпечення можна модифікувати під Unix-системи, адже дані системи є більш надійними і їх все більше використовують для серверних комп'ютерів. Причому в першу чергу повинен бути модифікований модуль Агент, адже саме він повинен працювати на сервері з RAID-масивом.

Код програми повинен бути написаний з урахуванням того, щоб у майбутньому було не складним його застосувати для написання такого ж модуля для Unix-платформи.

У якості середовища для розробки було обране середовище MS Visual Studio на мові C++ з кількох причин:

- простота і надійність створення і відлагодження програми;

- використання всіх переваг сучасних операційних систем сімейства Windows, включаючи багатозадачність, зручний інтерфейс та інше;
- наявність і доступність великої кількості компонент, що реалізують багато стандартних функцій;
- наявність потужних, зручних утиліт для відлагодження і тестування програмного забезпечення;
- наявність обширної системи довідкової інформації, посібників розробника, що полегшує створення програмного забезпечення;
- можливість адаптувати С-програмний код для переробки програми під Unix-систему.

Microsoft Visual Studio C++ – одне з найбільш поширених і найбільш ємних середовищ розробки. Багато розробників вважають Visual C++ найпотужнішою з усіх ІСР такого класу. Microsoft Visual Studio C++ являє собою набір з безлічі інструментів, зібраних у одному динамічному пакеті, готовому до негайної роботи (рис.3.5).

Рисунок 3.5. Відлагодження у середовищі Microsoft Visual Studio 2019 (C++)

Мова C++ дозволяє зберігати більшу частину програмного коду всередині самостійних об'єктів, а це скорочує обсяг великих програм. Крім цього, фірма Microsoft розробила бібліотеку Microsoft Foundation Classes. MFC – пакет, що складається з заздалегідь написаного і готового до роботи коду. Наприклад, замість того щоб самостійно писати програму для роботи з новим вікном, можна просто скористатися класом cWnd з MFC [2]. Можливості, що надаються бібліотекою класів MFC дозволяють конструювати елементи призначеного для користувача інтерфейсу, легко працювати зі стандартними типами даних мови С, розробляти класи, похідні від бібліотечних з додаванням нових функціональних можливостей, розробляти власні класи з подальшим розвитком їх функціональності. Також для розширення функціональності можна використовувати готові бібліотеки, які отримуються напряму з середовища розробки через вбудований у Visual Studio менеджер пакетів NuGet Package Manager.

Мова програмування C++ дозволяє розбивати програми на модулі, що потім можуть бути використані в інших програмах. C++ поставляється з великою кількістю стандартних бібліотек, які можна використовувати, як основу для нових програм або як приклади при вивченні мови. Стандартні модулі надають засоби для роботи з файлами, системними викликами, мережними з'єднаннями і навіть інтерфейсами до різних графічних бібліотек. C++ дозволяє писати зручні для читання програми завдяки загальноприйнятим узгодженням щодо написання коду та назв полів різних типів. Програми, написані мовою C++ звичайно значно коротші ніж їхні еквіваленти на С з декількох причин:

– типи даних високого рівня дозволять виразити складні операції однією інструкцією;

– наявність новітніх методів;

– широкий вибір методів та структур.

Синтаксис C++ близький до Java. Мова має строгу підтримує поліморфізм, наслідування, перевантаження операторів, інкапсуляцію, закриття методів, вказівники на функції та члени класів, атрибути, події, властивості, делегати,

винятки, коментарі у форматі XML.

Windows Forms дозволяє розробляти інтелектуальні клієнти. Інтелектуальний клієнт – це програма з повнофункціональним графічним інтерфейсом, просте в розгортанні і оновленні, здатне працювати при наявності або відсутності підключення до Інтернету і використовує більш безпечний доступ до ресурсів на локальному комп'ютері в порівнянні з традиційними додатками Windows. Windows Forms – це технологія інтелектуальних клієнтів для .NET Framework. Вона являє собою набір керованих бібліотек, що спрощують виконання стандартних завдань, таких як читання з файлової системи і запис в неї. При використанні середовища розробки, як Visual Studio, можна створювати інтелектуальні клієнтські програми Windows Forms, які відображають відомості, запитують введення від користувачів і обмінюються даними з віддаленими комп'ютерами по мережі. У Windows Forms, форма – це візуальна поверхня, на якій виводиться інформація для користувача. Зазвичай додаток Windows Forms будується шляхом приміщення елементів управління на форму і написання коду для реагування на дії користувача, такі як клацання миші або натискання клавіш. Елемент управління – це окремий елемент призначеного для користувача інтерфейсу, призначений для відображення або введення даних. При виконанні користувачем якої-небудь дії з формою або з одним з елементів управління створюється подія. Додаток реагує на ці події за допомогою коду і обробляє події при їх виникненні. Windows Forms включає широкий набір елементів управління, які можна додавати на форми: текстові поля, кнопки, списки, що розкриваються, перемикачі та навіть веб-сторінки. Якщо існуючий елемент управління не задовольняє потребам, в Windows Forms можна створювати власні елементи управління. До складу Windows Forms входять багатофункціональні елементи призначені для користувача інтерфейсу, що дозволяють відтворювати можливості таких складних додатків, як Microsoft Office. Використовуючи необхідні елементи управління, можна створювати панелі інструментів і меню, що містять текст і малюнки, та інші елементи управління, такі як текстові поля і поля зі списками.

За допомогою Visual Studio можна легко створювати додатки Windows Forms. Досить виділити елемент керування курсором і помістити його в потрібне місце на формі. Для подолання труднощів, пов'язаних з вирівнюванням елементів управління, конструктор надає такі додаткові елементи, як лінії сітки і лінії прив'язки. За допомогою Visual Studio або компіляції з командного рядка, можна використовувати елементи управління для створення складних макетів форм за менший час. У багатьох додатках потрібно відображати дані з бази даних, XML-файлу, веб-служби XML або іншого джерела даних. Windows Forms надає гнучкий елемент управління для відображення таких табличних даних в традиційному форматі рядків і стовпців так, що кожен фрагмент даних займає свою власну клітинку. За його допомогою можна, налаштувати зовнішній вигляд окремих осередків, зафіксувати рядки і стовпці на своєму місці, а також забезпечити відображення складних елементів управління всередині осередків. За допомогою Windows Forms можна легко створювати елементи управління з прив'язкою до даних. Створювати елементи управління з прив'язкою до даних можна шляхом перетягування об'єктів з допоміжного вікна в форми проекту. Також можна пов'язувати існуючі елементи управління з даними, перетягуючи об'єкти в існуючі елементи управління. Інший тип прив'язки до даних в формах Windows Forms – це параметри. Більшість інтелектуальних клієнтських додатків повинні зберігати деякі відомості про свій стан під час виконання, такі як відомі розміри форм, а також зберігати призначені для користувача дані, наприклад місце збереження файлів за замовчуванням. Додаток Windows Forms надає простий спосіб зберігання обох типів відомостей на клієнтському комп'ютері. Після визначення ці параметри за допомогою Visual Studio або редактора коду, зберігаються в XML-файлі і автоматично зчитуються в пам'ять під час виконання.

Програмний модуль Агент складається з 14 класів. У Додатку А наведений лістинг декількох найбільш важливих класів.

4 ОПИС І ЕКСПЛУАТАЦІЯ ПЗ ДЛЯ АДМІНІСТРУВАННЯ НАДЛИШКОВОГО ДИСКОВОГО МАСИВУ

4.1 Інтерфейс та можливості програмного модуля Агент

Програмний модуль Агент складається з двох окремих інтерфейсів. Перший служить для пошуку підключених RAID-контролерів на даному комп'ютері. Другий – безпосередньо сам сервер, який відповідає за зв'язок модуля Менеджер і надлишкового дискового масиву, а також за нотифікацію в разі несправності.

4.1.1 Реалізація модуля для пошуку підключених RAID-контролерів

Модуль реалізований окремо від серверної частини, тому що він буде працювати тільки в момент первинного завантаження сервера. Його безпосереднє завдання – просканувати підключені RAID-контролери до USB-порту. Якщо сканування пройшло успішно, необхідно записати файл налаштувань для сервера, в якому буде інформація про підключений RAID-контролер.

Модуль працює у діалоговому режимі. Кожне діалогове вікно містить певний набір кнопок для управління процесом встановлення підключення (рис. 4.1–4.5). Коротко пояснимо призначення основних типів кнопок:

- «Відмінити» – скасування дій;
- «Закінчити» – завершення роботи пошуку;
- «<<Назад» – перехід на крок назад;
- «Далі >>» – перехід на крок вперед.

Модуль виконаний так, що користувач покроково проходить всі вікна налаштувань і в кінці отримує результат про знайдені RAID-пристрої.

Рисунок 4.1. Інтерфейс вибору типу підключення RAID-масиву

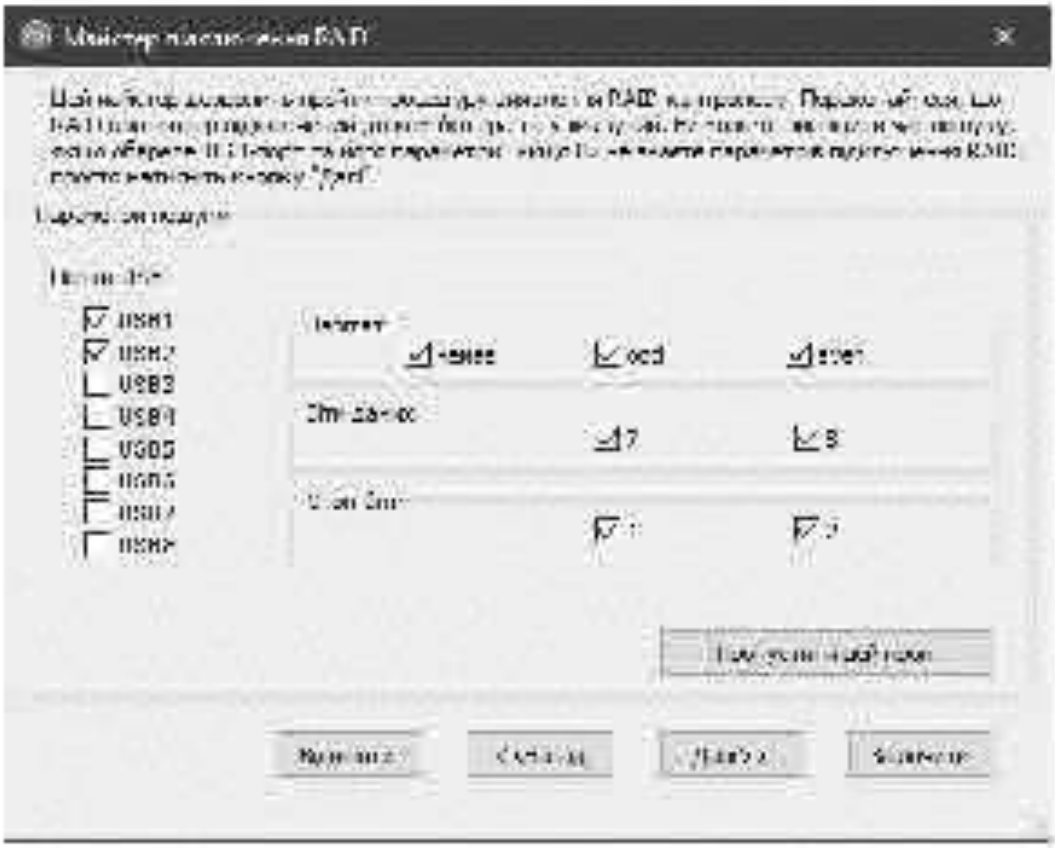

Рисунок 4.2. Інтерфейси налаштування параметрів пошуку

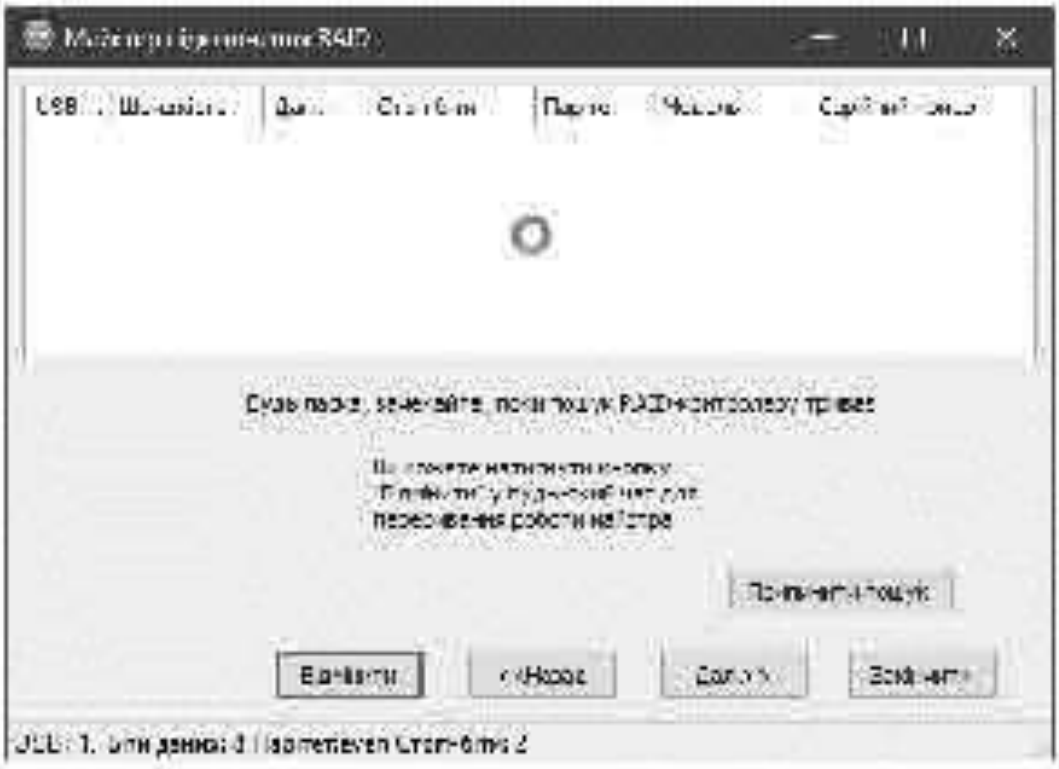

Рисунок 4.3. Інтерфейс пошуку підключених RAID-масивів

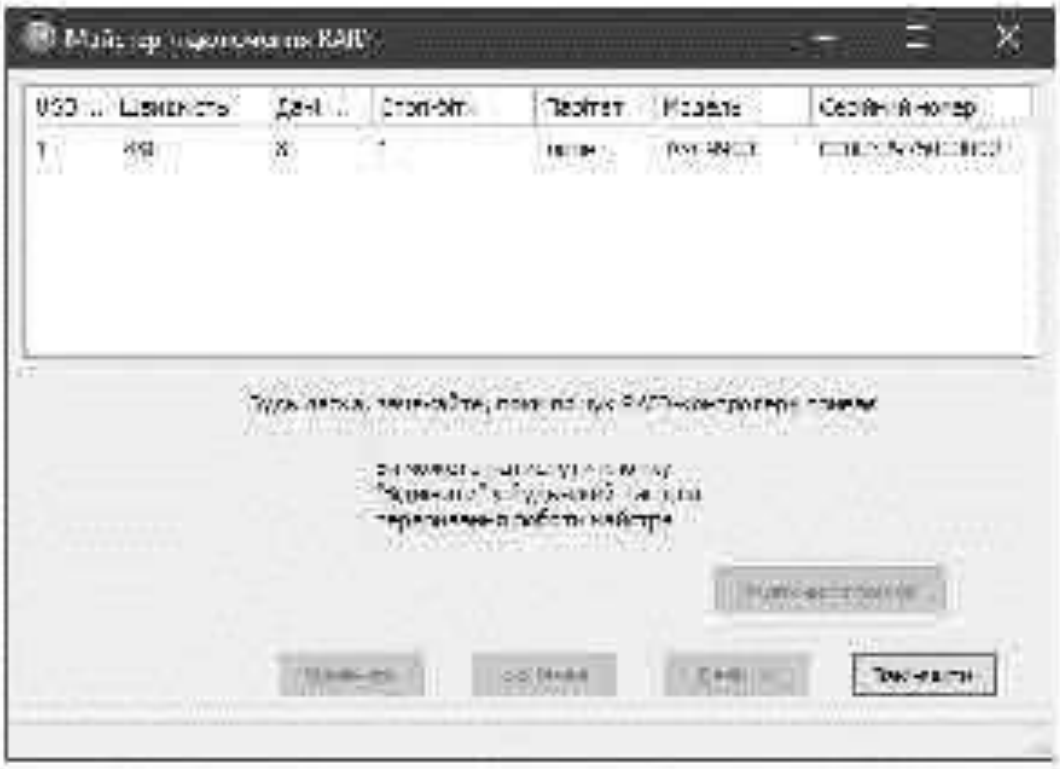

Рисунок 4.4. Інтерфейс пошуку при знаходженні RAID-масиву

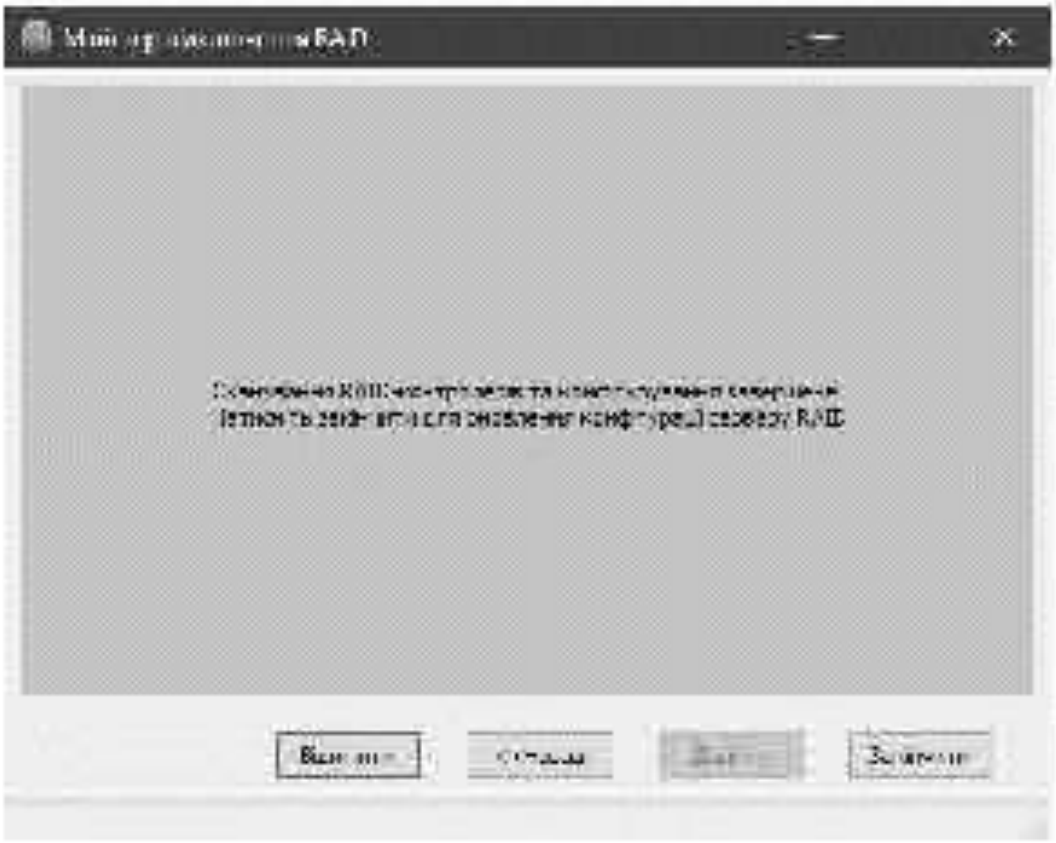

Рисунок 4.5. Інтерфейс закінчення роботи майстра підключення

У будь-який момент він може повернутися на один або кілька кроків назад, якщо якась інформація, введена ним, є невірною.

Критерієм знаходження контролера є:

- Успішне з'єднання по інтерфейсу USB;
- Відповідь на команду "GetPage 0;";
- Наявність в перших трьох байтах відповіді сигнатури «ACS».

Після успішного пошуку система запропонує зберегти налаштування для сервера. Якщо файл вже існує, буде видано попередження про перезапис цього файлу.

4.1.2 Реалізація сервера Агент

Після установки серверної частини програми програмний модуль Агент готовий до запуску. При першому запуску, а також якщо файлу налаштувань раптом було не виявлено, система запропонує скористатися модулем для пошуку підключених RAID-контролерів.

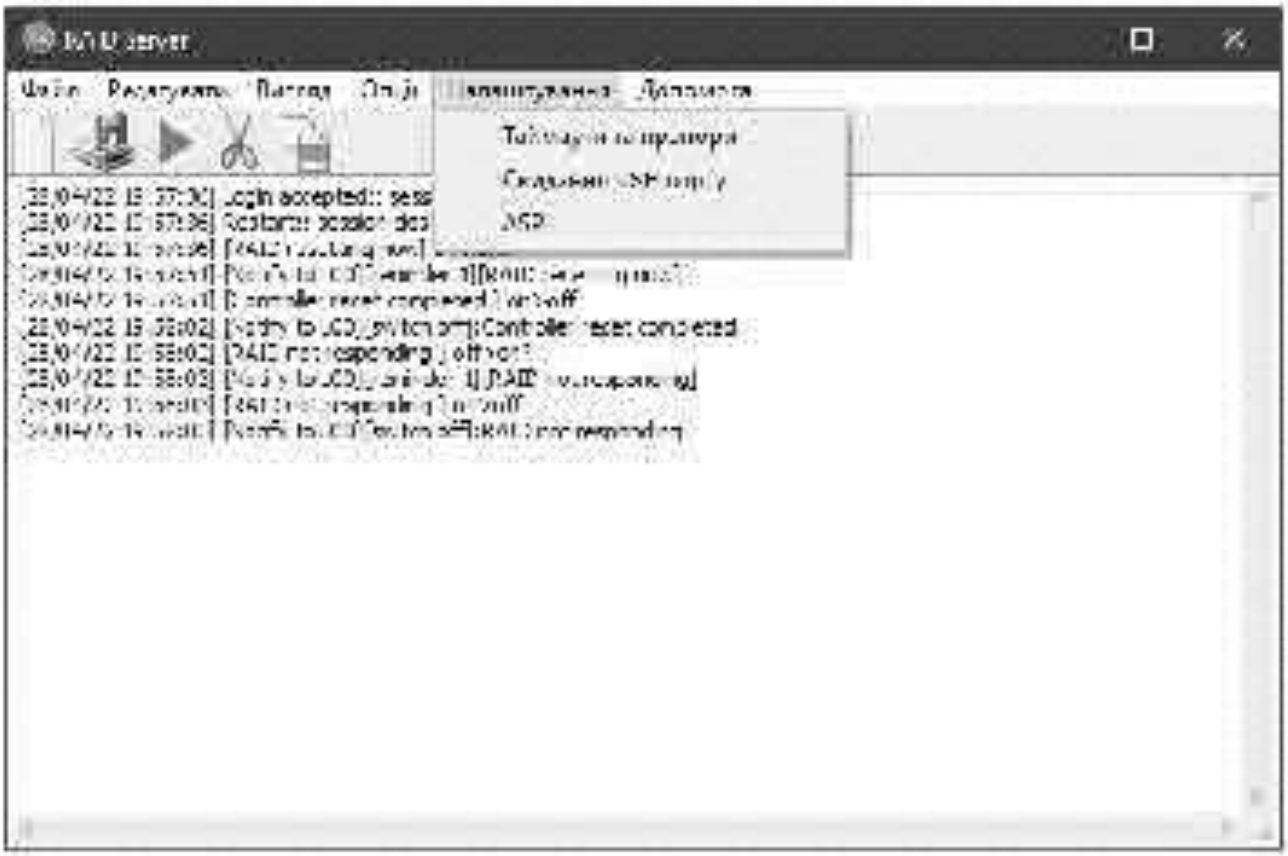

Рисунок 4.6. Головне вікно модуля Агент

Цей же модуль запускається, якщо RAID-контролер з налаштуваннями, зазначеними у файлі agent.ini, не був знайдений.

При нормальному запуску програма не відкриває ніяких вікон, а просто поміщає свою іконку в Trayбar (місце в панелі Windows праворуч). При згортанні вікна, вона також ховається у Trayбar. Якщо натиснути на іконку програми правою кнопкою миші, спливе меню для швидких налаштувань.

При подвійному натисканні мишею по іконці спливе основне вікно, в якому можна стежити за роботою Агента. Дане вікно показано на рис. 4.6. Всі дії, які виконуються даним модулем, відображуються на екрані у вигляді текстових повідомлень формату log-файлу. За допомогою даного екрану можна стежити за всіма діями Агента. При бажанні можна тимчасово відключити виведення повідомлень на екран, хоча у log-файл ці повідомлення все одно потраплять.

Управляти програмою можна через меню або Toolbar (місце під меню, з іконками для швидкого запуску команд). Кнопки Toolbar дублюють відповідні позиції меню і тому далі розглядатись не будуть.

Розглянемо функції меню. Меню File надає сервіс відповідно до CUA-стандарту:

Файл|Новий – очищає екран від дамп-повідомлень;

Файл|Зберігти – дозволає зберегти дамп-повідомленняу файл;

Файл|Відправити – дозволяє відіслати дамп-повідомлення через MS Exchange;

Файл|Вихід – дозволяє завершити роботу Агенту. Аналогічне натисканню на кнопку у правому верхньому куті вікна. При виході система запитає підтвердження на завершення роботи Агенту (рис. 4.7).

Рисунок 4.7. Запит на підтвердження виходу

Меню Edit також надає сервіс відповідно до CUA-стандарту:

Редагувати|Вирізати, Редагувати|Копіювати – надають користувачеві можливість зберігати дамп у Clipboard.

Меню Вигляд надає можливість управляти toolbar`ом і statusbar`ом:

Вигляд|Toolbar – управляє видимістю toolbar`у;

Вигляд|Status Bar – управляє видимістю statusbar`у.

Меню Опції надає можливість управляти дампом, а також перевіряти працездатність системи. Даний пункт меню створювався саме для тестування системи.

ОпційТестування – при виборі цієї позиції меню відображається немодальне вікно, за допомогою якого можна імітувати деякі аварійні ситуації (рис. 4.8);

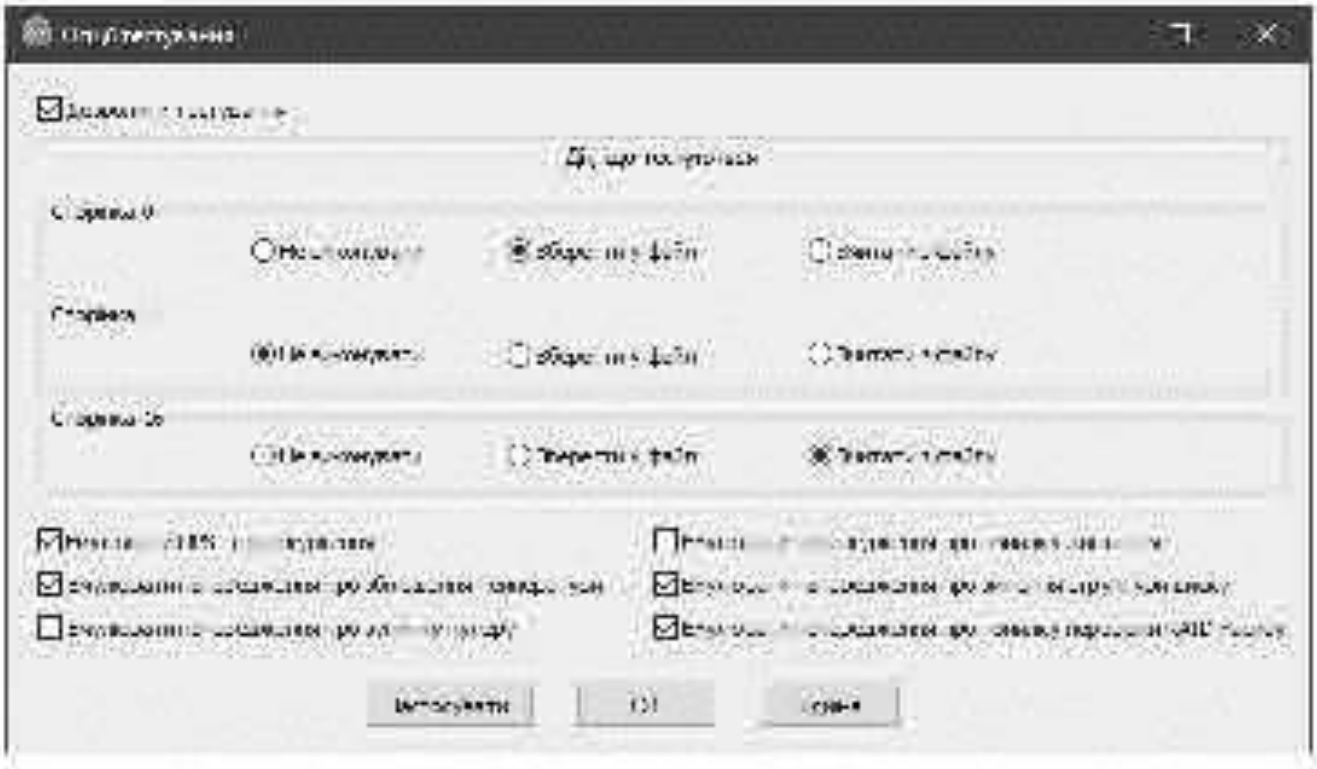

Рисунок 4.8. Вікно налаштувань тестування

Використовуючи багатопозиційні перемикачі (Radiobuttons), адміністратор може записати інформаційні сторінки RAID-контролера на диск, змінити їх і замінити ними реальні сторінки контролера. Таким чином, можна імітувати практично будь-яку ситуацію. Ще простіше – змінити стан прапорців (CheckBox). В інформаційних сторінках RAID примусово встановлюються відповідні біти і можна простежити роботу нотифікації і працездатності GUI.

ОпційТестувати почту – дозволяє перевірити відсилання e-mail, повідомлення "Тестування пошти" відсилається всім прописаним у ini-файлі e-mail клієнтам;

ОпційТестувати MS net – дозволяє перевірити оповіщення через MS Network messaging. Повідомлення "Тестування мережі" відсилається всім прописаним у ini-файлі network клієнтам;

ОпціїЛогування – управляє видачою дампу у головне вікно. Якщо ця позиція меню відзначена, то команди, оброблювані Агентом, пишуть повідомлення у головне меню програми.

4.1.3 Загальний вигляд модуля Менеджер

Даний модуль розроблявся паралельно з розробкою модуля Агент. Тестування працездатності проходило за його безпосередньої участі, адже одне із завдань Агенту полягає в тому, щоб зв'язати роботу Менеджеру і RAID-контролеру. На рис. 4.9 показаний загальний вигляд модуля Менеджер. При цьому також працює модуль Агент.

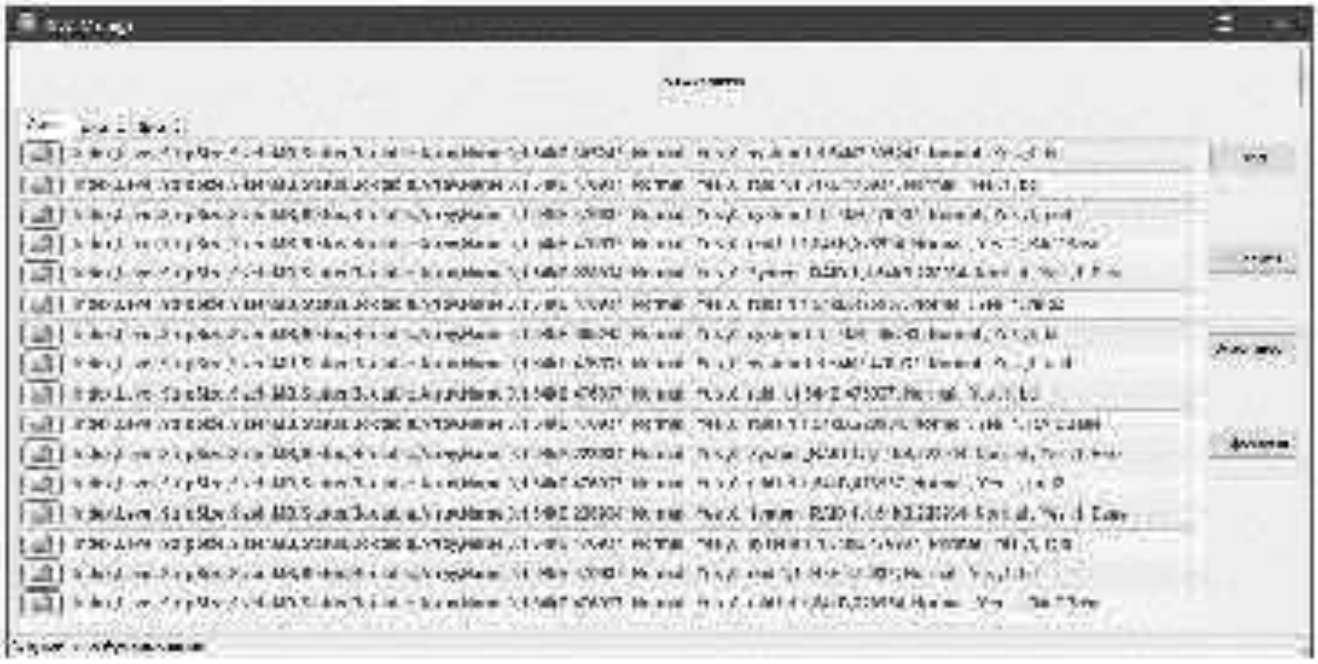

Рисунок 4.9. Загальний вигляд модулю Менеджер

4.2 Встановлення розробленого ПЗ на комп'ютер користувача

Для встановлення програмного забезпечення на комп'ютер користувача, необхідно запустити файл setup.exe, що знаходиться у дистрибутиві. Програма встановлення (інсталятор) не відрізняється від подібних програм, оскільки була створена за допомогою популярної програми InstallShield і з результатами роботи цієї програми стикався практично кожен користувач (рис. 4.10). За замовчуванням програмне забезпечення віддаленого керування RAID-масивом

встановлюється в каталог C:\Program Files\RAID-MAN\, проте при установці користувач має можливість вибрати інший каталог.

Рисунок 4.10. Встановлення програмного забезпечення GUIMAN

При установці можна встановлювати або обидва модуля (Агент і Менеджер) відразу, або кожен з них окремо.

4.3 Методика випробувань програми і результати експериментальної перевірки

Тестування – один з ключових елементів забезпечення якості. Багато відомих розробники ПЗ проводять тестування своїх продуктів в кілька етапів, які відрізняються видами виконуваних робіт і ресурсами, що залучаються. Останнім часом у зв'язку зі створенням великих програмних систем зріс інтерес до методики розробки і, зокрема, відлагодження програм.

Методика розробки та відлагодження програмних систем повинна доповнюватися і методикою виготовлення і відлагодження окремих програмних блоків, підпрограм, модулів, що розробляються одним програмістом. Без застосування ефективних способів створення таких програмних одиниць не

можна сподіватися успішно вирішити і проблему створення програмних комплексів.

Проблема відлагодження існує також і для програм середньої складності. Для таких програм ефективність і достовірність відлагодження не є настільки життєво необхідним, і виявлення серйозних помилок в ході експлуатації програми не призводить до настільки сумних наслідків, як для великих систем, адже автор програми зазвичай буває в змозі виправити їх в прийнятні терміни.

4.3.1 Відлагодження і загальні принципи тестування

Одним з найбільш складних і трудомістких етапів технологічного процесу розробки програм є їх тестування і відлагодження. Як відомо, при створенні типового програмного проекту близько 50% загального часу і більше 50% загальної вартості витрачається на перевірку (тестування) програми, що розробляється, і її відлагодження.

Під тестуванням слід розуміти процес виконання програми з метою виявлення помилок, в якості яких приймається будь-яке відхилення від еталонів. Добрим вважається тест, який має високу ймовірність виявлення ще не знайдених помилок.

Під відлагодженням розуміється процес, що дозволяє отримати програму, функціонуючу з необхідними характеристиками в заданій області вхідних даних. Таким чином, в результаті відлагодження програма повинна відповідати деякій фіксованій сукупності правил і показників якості, прийнятої за еталонну для даної програми.

Ефективність тестування є найважливішим чинником, що визначає вартість і тривалість розробки складних комплексів програм з заданою якістю. Внаслідок цього створюються різні методи тестування, що забезпечують найкраще використання ресурсів проектування.

Фактично, тестування починається ще в процесі кодування чергової версії: оперативно тестуються і відлагоджуються знову розроблювані або змінювані функції системи. Подібна організація робіт дозволяє заощадити час і сили,

оскільки значна частина помилок виявляється і усувається практично в момент виникнення. Робота тестувальника (відладчика) на цьому етапі як би локалізована в рамках одного модуля або частини системи, що розробляється, даною групою (для великих систем), тому таке тестування називається локальним.

При певній мірі готовності системи проводиться перехресне тестування: розробники різних частин «свіжим поглядом» перевіряють роботу один одного, одночасно обмінюючись досвідом. При створенні невеликих програмних продуктів, коли програміст-розробник одночасно є тестувальником, цей етап зазвичай пропускається, часто даремно.

І локальне і перехресне тестування супроводжується перевіркою вихідного коду. Якщо робота тестувальника з системою – це пошук помилок по їх проявам в процесі виконання програми, то робота з вихідним кодом дозволяє «відловити» помилки, які при звичайному тестуванні виявляться не відразу.

Існує два основні підходи до тестування:

1. Тестування програми як "чорного ящика". Програма розглядається як чорний ящик. Тестові дані використовуються тільки відповідно до специфікації програми, тобто без урахування знань про її внутрішню структуру. При такому підході виявлення всіх помилок в програмі є критерієм вичерпного вхідного тестування. Останнє може бути досягнуто, якщо в якості тестових наборів використовувати всі можливі набори вхідних даних. В цьому випадку для вичерпного тестування потрібні нескінченні набори тестів. Однак існують певні методології тестування, що дозволяють з усіх можливих тестових наборів виділити деяку підмножину тестів, що мають найвищу ймовірність виявлення більшості помилок.

2. Тестування програми як "білого ящика". Стратегія білого ящика, або стратегія тестування, керованого логікою програми, дозволяє використовувати внутрішню структуру програми. В цьому випадку програміст отримує тестові дані шляхом аналізу логіки програми.

Також існує 3 основних способи тестування: алгоритмічне, аналітичне і змістовне.

Алгоритмічне тестування застосовується програмістом для контролю етапів алгоритмізації і програмування. Програмісти проектують тести і починають готувати еталонні результати на етапі алгоритмізації, а використовують їх на етапі відлагодження.

Аналітичне тестування служить для контролю обраного методу розв'язання задачі, правильності його роботи в обраних режимах і з встановленими діапазонами даних. Тести проектують і починають готувати відразу після вибору методу, а використовують їх на останньому етапі відлагодження або для аналізу результатів пробного рахунку; в ході тестування, поряд зі звіркою на збіг, застосовуються і якісні оцінки результатів.

Змістовне тестування служить для перевірки правильності постановки завдання. Для контролю при цьому використовуються, як правило, якісні оцінки і статистичні характеристики програми, фізичний зміст отриманих результатів і т.п. У проведенні змістовного тестування, принципи якого формулюються в технічному завданні, найактивнішу участь повинні брати замовники або користувачі програми.

Змістовні та аналітичні тести перевіряють правильність роботи програми в цілому або великих її частин, в той час, як алгоритмічні тести в першу чергу повинні перевіряти роботу окремих блоків або операторів програми.

4.3.2 Методика оцінювання результатів тестування програми

При тестуванні розроблюваного програмного модуля, в основному застосовувалися такі способи відлагодження:

1) Відлагодження за допомогою вбудованого відладчика.

У середовище програмування MS Visual Studio вбудований потужний і зручний у використанні відладчик. З його допомогою можливо відстежувати значення довільної кількості змінних в процесі роботи програми, здійснювати

порядкове виконання коду, встановлювати точки зупинки програми і виходу в режим відлагодження;

2) Методи силового відлагодження.

Під силовим відлагодженням (brute-force debugging) розуміються методи відлагодження, засновані не на можливостях відладчиків. При розробці програм часто немає необхідності в повному відлагодженні, просто треба переконатися в тому, що будь-яка функція працює так, а не інакше. У цих випадках простіше додати необхідні рядки коду для виведення інформації. Методи "силового відлагодження" були ефективні при відлагодженні і тестуванні даної програми і використовувалися нарівні із засобами відладчику MS Visual Studio;

3) Виведення відлагоджувальної інформації на формі.

Один із способів виведення інформації – її виведення безпосередньо в формі. Зазвичай найпростіше створити форму у редакторі ресурсів MS Visual Studio для безпосереднього виведення інформації. В такому випадку виведена інформація не загубиться навіть при перемальовуванні форми;

4) Використання функції AfxShowMessage .

Крім виведення інформації в форму, можна скористатися модальним діалоговим вікном. Принципова відмінність цього методу, в першу чергу, полягає в тому, що модальне діалогове вікно зупиняє виконання програми, поки його не закриють. Таким чином, у розробника є достатньо часу, щоб прочитати і осмислити отриману інформацію. Функція Afx ShowMessage ідеально підходить для цієї мети. Вона дозволяє вивести рядок будь-якої довжини в простому модальному діалоговому вікні. Потрібно тільки створити рядок для виведення і передати його у функцію;

5) Виведення на консоль і запис у лог-файл.

До методів силового відлагодження також відносяться "Виведення відлагоджувальної інформації на консоль" і "Запис даних у log-файл". У першому випадку створюється консольний додаток і виведення до нього здійснюється за допомогою процедур cout і printf. У іншому випадку вся інформація записується у файл на диску. В даному проекті інформація, отримана

з log-файлу, була найцінішою, оскільки точно вказувала на місце виникнення помилок і неточностей у роботі алгоритмів.

Представлений програмний модуль Агент в складі проекту RAID Manager відповідає за постійну працездатність надлишкового дискового масиву. Тому даний модуль повинен постійно працювати на комп'ютері з надлишковим дисковим масивом. З цієї причини повинна бути забезпечена висока надійність цього модуля, для того щоб він не переставав працювати з вини критичної помилки.

Виходячи з цього, можна виділити головні чинники, що впливають як на надійність даного програмного модуля, так і на надійність будь-якого програмного забезпечення:

1) коректність структури програми;

2) коректність обробки даних;

3) стійкість до помилкового введення даних користувачем.

Стійкість до помилкового введення даних користувачем реалізована при перевірці заповнення різних форм. В основному на коректність форми перевіряються ще при заповненні в модулі Менеджер перед відправкою їх Агенту. Але для надійності перевірка на правильне заповнення полів існує і в модулі Агент. В першу чергу це стосується значень параметрів, які передаються при запиті змінити параметри системи.

У разі введення невірних даних Агент повертає модулю Менеджер код команди помилки і саме повідомлення про помилку. Менеджер, в свою чергу, це повідомлення про помилку видає користувачеві на екран.

4.4 Тестування працездатності RAID-контролера

Спеціально для тестування працездатності RAID-контролера було розроблено додаток, за допомогою якого можна було відправляти команди для RAID і дивитися отримані відповіді. Загальний вигляд цього додатка показаний на рис. 4.11.

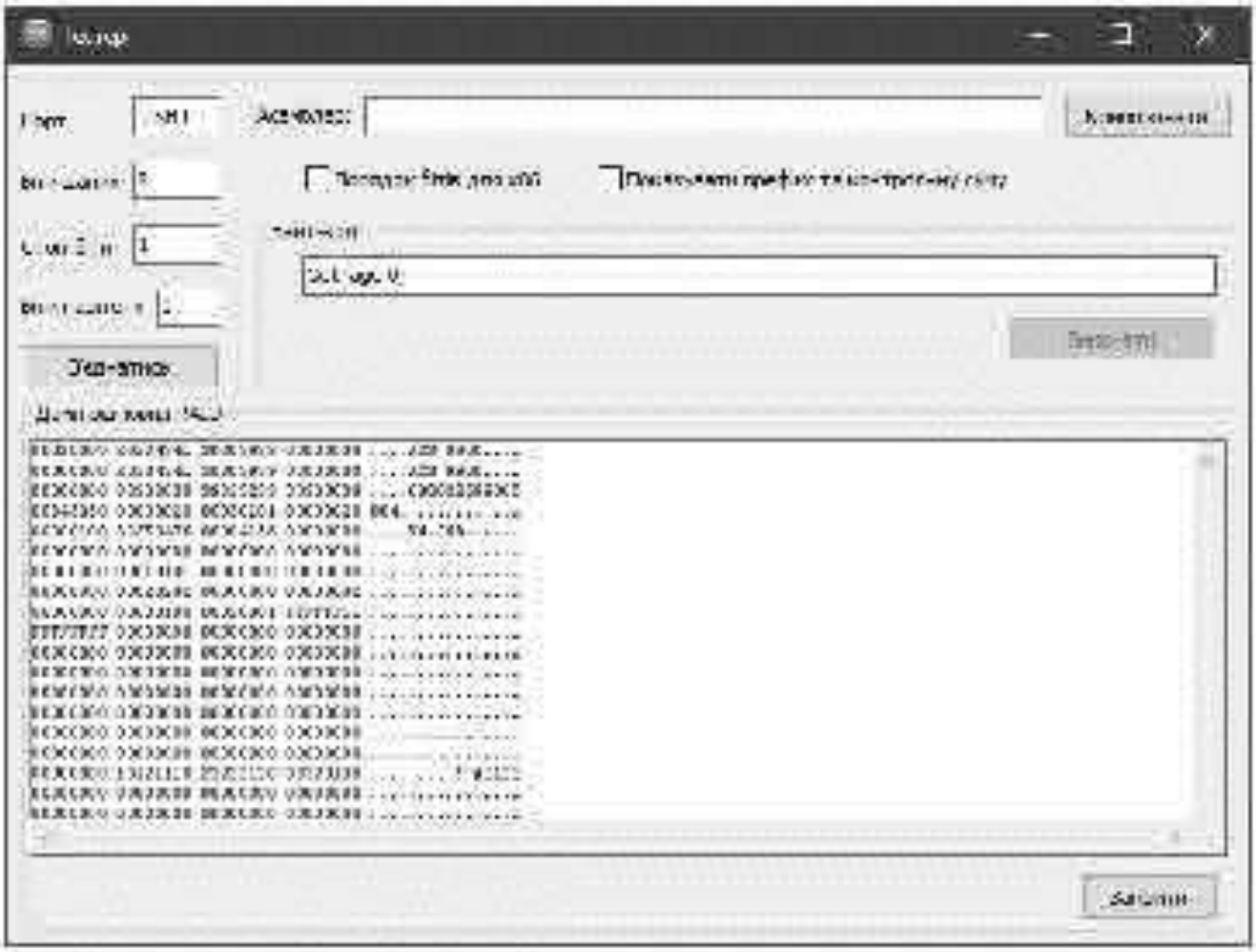

Рисунок 4.11. Загальний вигляд інтерфейсу додатку Tester

Цей додаток значно полегшив написання коду модуля Агент, адже було видно, які відповіді отримує Агент на свої запити.

4.5 Висновки за розділом

В даному розділі були розглянуті питання розробки, відлагодження, тестування і надійності програмних продуктів. Було наведено обґрунтування необхідності і важливості етапу відлагодження в процесі розробки програмного забезпечення, дані короткі описи основних способів відлагодження і тестування. Розглянуто критерії надійності систем, основні чинники, що впливають на надійність функціонування комплексів програм, проведено аналіз надійності програмних продуктів. Представлені результати роботи програми.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Резюме

В даному дипломному проекті розроблене програмне забезпечення для адміністрування надлишкового дискового масиву, що дозволяє користувачеві стежити за його роботою і віддалено налаштовувати з будь-якого комп'ютера, не вимагаючи при цьому спеціальних знань з керування RAID-масивом. Оповіщення адміністратора при виявлення будь-яких помилок роботи надлишкового дискового масиву дозволить оперативно реагувати на них.

5.2 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку. Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога.

Таблиця 5.1 Каталог аналогів

Найменування ПП	Обсяг функції ПП – V_o, усл. машинних командах.
1. ПП автоматизації засобів по каталогу	680 – 7000
2. ПП автоматизованих розражунків	1300 – 8600
3. ПП загальної математики і ПП імітаційного моделювання	7800 – 8800

У таблиці 5.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт. Для нашого варіанта виділено сірим кольором.

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл 5.2

Таблиця 5.2

Обсяг ПП, тис умов машинних команд	Норма часу, люд/год
1.00	229
2.00	244
3.00	262

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): Тар = 229 х 0,7 = 160,3 (люд/годин).

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

$$T_{тз} = T_{пр} \times L_1 \times K_н \quad (5.1)$$

$$T_{тп} = T_{пр} \times L_2 \times K_н \quad (5.2)$$

$$T_{рп} = T_{пр} \times L_3 \times K_н \times K_т \quad (5.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 5.2.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 5.3.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 5.4.).

Таблиця 5.2 Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Для нашого варіанта виділено сірим кольором.

Таблиця 5.3 Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	Ступінь новизни	Значення $K_н$
А	Принципово нові ПО	1,75 – 1,2
Б	ПО – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПО аналогічний аналог	0,7

Для нашого варіанта виділено сірим кольором.

Таблиця 5.4 Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблюваного ПО типовими програмами, %	Значення $K_т$
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Для нашого варіанта виділено сірим кольором.

Тепер розрахуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$T_{тз} = Та * L_1 * K_п = 160,3 * 0,12 * 0,8 = 15,38 \text{ (людн/годин)} \quad (5.1)$$

Трудомісткість розробки технічного проекту

$$T_{тп} = Та * L_2 * K_п = 160,3 * 0,11 * 0,8 = 14,11 \text{ (людн/годин)} \quad (5.2)$$

Трудомісткість розробки робочого проекту

$$T_{рп} = Та * L_3 * K_п * K_т = 160,3 * 0,61 * 0,8 * 0,8 = 62,58 \text{ (людн/годин)} \quad (5.3)$$

Для подальших розрахунків визначили кількість паперу, витраченого на кожен етап: технічне завдання $N_{тз}$=3 (стр), розробка ТП $N_{тп}$=9(стр), розробка робочого проекту $N_{рп}$=14 (стр),пояснювальна записка відповідно $N_{пз}$=36 (стр)

Розрахунок зведений у таблицю 5.5

Таблиця 5.5 Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{рТЗ}$=15,38	$T_{кк}$=0,7·$N_{тз}$= 0,7*3=2,1	$T_{пк}$=0,15·$N_{тз}$=0,15*3=0,45
2 Розробка ТП	$T_{рТП}$=14,11	$T_{кк}$=0,7·$N_{тп}$=0,7*9=6,3	$T_{пк}$=0,15·$N_{тп}$=0,15*9=1.4
3 Розробка РП	$T_{рРП}$= 62,58	$T_{кк}$=0,7·$N_{рп}$=0,7*14=9,8	$T_{пк}$=0,15·$N_{рп}$=0,15*14=2,1
4 Розробка ПЗ	$T_{пз}$=1,5·$N_{пз}$= 1,5*36 =54	$T_{кк}$=0,7·$N_{тз}$=0,7*36=25,2	$T_{пк}$=0,15·$N_{пз}$=0,15*36 =5,4
Усього, в т.ч.:	198,9		
- на розробку	$\Sigma T_р$=146.1		
- контроль керівника		$\Sigma T_{кк}$=43,4	
-нормоконтроль			$\Sigma T_{пк}$=9,4

5.3 Розрахунок ціни програмного продукту

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПО. Розрахунок основної заробітної плати виконавців приведений у таблиці 5.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2021» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2022 року - 6500 гривень; мінімальну погодинну тарифну ставку – 39.26 грн.

Таблиця 5.6 Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1 Розробка ПП	146,1	39,26	5735,89
2 Контроль керівника	43,4	60,10	2608,34
3 Нормоконтроль	9,4	60,10	564,94
Усього	-	-	ΣЗо= 8909,17

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 5.7

Таблиця 5.7 Розрахунок матеріальних витрат на розробку ПО

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці грн.	Вартість, грн.
Папір	Лист А4	70	2.0	140,0
Разом	-	-	-	$В_{м1}$=140,0
Транспортно – заготівельні Витрати (10%)				$В_{тр\_з} = 0,1 \times В_{м1}$= 0,1*140=14,00
Усього				$В_м = В_{м1} + В_{тр\_з}$ - 154.00

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 5.8.

Таблиця 5.8 Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	154.00	$В_м$ (див. табл. 5.7)
2. Основна заробітна плата	8909,17	$З_о$ (див. табл. 5.6)
3. Додаткова заробітна плата	890,92	$З_д = 0,1 \times З_о$ = 8909,17*0,1
4 Відрахування до єдиного фонду соціального внеску	2156,02	$В_{с.с.в.} = 0,22 \times (З_о + З_д)$=0,22* (8909,17+890,92)
5. Накладні витрати	3563,67	$В_{нак.} = 0,4 \times З_о$=0,4 *8909,17
6. Повна собівартість	15673,78	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +$В_{с.с.в.}$ + $В_{нак.}$= 154.00+8909,17+890,92+2156,02+ 3563,67

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$$П=(С_п*Р)/100=(15673,78*10)/100=1567,38 грн \qquad (5.4)$$

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (контрорисна вартість) визначається по формулі:

$$Ц_п = С_п + П = 15673,78+1567,38= 17241,16 грн; \qquad (5.5)$$

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$$Ц_р = Ц_п + ПДВ = 17241,16 + 17241,16 *0.2 = = 20689,39 грн; \qquad (5.6)$$

6 ОХОРОНА ПРАЦІ

Власник або уповноважений ним органи зобов'язані дбати про умови праці працівників, полегшувати їх, оздоровляти навколишнє середовище, дбати про виконання правил безпеки і інструкцій по техніці безпеки. Координує всю цю діяльність служба охорони праці, яка в залежності від чисельності працюючих може функціонувати як самостійний структурний підрозділ (число працюючих 50 і більше), або у вигляді групи спеціалістів чи одного спеціаліста, у тому числі за сумісництвом (число працюючих 20 іменше). Задачі службі охорони праці та її функції викладені в Типовому положенні про службу охорони праці», яке затверджено наказом Комітетом Держнаглядохоронпраці (ДНАОП 0.00-4.21-93)

Охорона праці - це система забезпечення здоров'я і безпеки життя людини в проміжок часу застосування його праці, виступає однією з умов хорошого життя і забезпечення людини.

Повне дотримання всіх вимог системи охорони праці може певною мірою гарантувати безпеку всіх працівників на підприємстві. Норми права спрямовані також на регулювання відносин у сфері охорони праці між роботодавцем і працівником.

Відповідно до теми дипломної роботи, у цьому розділі розглядається безпека праці при розробці алгоритмічного та програмного забезпечення для адміністрування надлишкового дискового масиву.

В розділі охорона праці дипломного проекту розглядається питання охорони праці програміста на стадії вирішення таких питань розробки.

6.1 Аналіз небезпечних та шкідливих чинників при роботі з ПК

Аналіз умов праці, технологічних процесів, апаратури і обладнання з точки зору можливості виникнення появи небезпечних факторів, виділення шкідливих виробничих речовин. На основі такого аналізу визначаються небезпечні ділянки виробництва, можливі аварійні ситуації, розробляються заходи щодо їх усунення або обмеження наслідків.

6.2 Розробка заходів з охорони праці

Вибір технічних засобів забезпечення безпеки повинен здійснюватися на основі вивчення особливостей кожного виявленого небезпечного й шкідливого виробничого фактора і зони його дії – так званої небезпечної зони.

Можливі відхилення параметрів мікроклімату, виробничого освітлення, вентиляція приміщення, шуму.

6.2.1 Виробничі приміщення

Площа приміщення на одну людину по вимогам ДСанПіН 3.3.2-007-98 становить 6м квадратних, а об'єм 20м кубічних, в приміщенні є вікна. Вони орієнтовані на південь – стіни світло-блакитного кольору; підлога-зеленого;

Висота приміщень повинна бути не менше 3,2 м, складських приміщень – 3,0 м. Стіни повинні бути пофарбовані матовою фарбою. Полиу всіх приміщеннях повинні бути рівними,неслизькими, без щілин і бавор, зручними для санітарного мокрого і сухого прибирання.

6.2.2 Мікроклімат виробничого приміщення

Основними нормативними документами, що регламентують параметри мікроклімату виробничих приміщень, є ДСН 3.3.6.042-99 та ГОСТ 12.1.005-88. Ці параметри нормуються для робочої зони - визначеного простору, в якому знаходяться робочімісця.

У нормативному документі ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» параметри мікроклімату повинні відповідати даним таблиці 1.

Таблиця 1 – Параметри мікроклімату приміщення

Період року	Категорія робіт	Температура, С		Відносит. Вологість, %
		оптимальна	допустима	
Холодний	Легка – Іа	22-24	21-25	40-60
Теплий	Легка – Іа	23-25	22-26	40-60

Іа – категорія легких фізичних робіт при яких витрата енергії дорівнює 105-140 Вт (90-120 ккал/год)

У приміщенні де працює оператор параметри мікроклімату відповідають нормам з таблиці 1.

6.2.3 Вентиляція приміщення

Для створення в приміщенні нормальних умов мікроклімату для працівника і видалення шкідливих забруднень, була спроектована і належним чином встановлена вентиляційна система – загально обмінна, припливно-витяжна по нормам ДСТУ Б А.3.2-12:2009 ССБП.

Вентиляція створює на робочому місці, метеорологічні умови і чистоту повітряного середовища, що відповідають чинним санітарним нормам. Разом з тим вентиляція забезпечує умови, що відповідають вимогам технологічного процесу.

Дипломним проектом передбачена вентиляція у всіх виробничих та допоміжних приміщеннях. Це змішана вентиляція – природна та механізована.

6.2.4 Освітлення робочого місця, шум, вібрація

Для створення сприятливих умов для здорової роботи, які б запобігали швидкій втомлюваності очей, виникненню професійних захворювань, нещасних випадків і сприяли підвищенню продуктивності праці та якості продукції, виробниче освітлення повинно відповідати наступним вимогам:

- створювати на робочій поверхні освітленість, що відповідає характеру зорової роботи і не є нижчою за встановлені норми;
- забезпечити достатню рівномірність та постійність рівня освітленості у виробничих приміщеннях, щоб уникнути частої переадаптації органів зору;
- не створювати засліплювальної дії як від самих джерел освітлення, так і від інших предметів, що знаходяться в полі зору;
- не створювати на робочій поверхні різких та глибоких тіней (особливо рухомих);
- повинен бути достатній для розрізнення деталей контраст поверхонь, що освітлюються.

❖ не створювати небезпечних та шкідливих виробничих чинників (шум, теплові випромінювання, небезпека ураження струмом, пожежо- та вибухонебезпека світильників);

❖ повинно бути надійним і простим в експлуатації, економічним та естетичним.

6.2.5 Шум

Так як шум має 35 Дб, сприйняття шуму людським вухом межується від 20дб до 120 дб, це означає, що при роботі за ЕОМ шум не заважає, працівнику працювати.

Для запобігання виникнення інших шумів у відповідності з ГОСТ 12.1.029-80 зниження шуму й вібрації в приміщенні дипломним проектом передбачені звукоізоляція віконта дверей.

6.3 Пожежна безпека

Забезпечення пожежної безпеки об'єкта здійснюють на основі вимог, а також інших нормативних документів .

Основними напрямами забезпечення пожежної безпеки є усунення умов виникнення пожежі та мінімізація її наслідків.

Правильний вибір пожежної сигналізації - шлях до максимальної безпеки будинку, квартири та офісу.

Основа системи - пожежний сповіщувач. Це невеликий чутливий датчик, моментально фіксує осередки займання. Аналізуючи зібрану ним інформацію, пожежникам та власникам будівель вдається запобігти поширенню вогню, встановити контроль над ситуацією, що склалася.

Сучасні сповіщувачі - запорука безвідмовної роботи систем сигналізації

✓ *Теплові*. Сповіщувач реагує на температурні перепади

✓ *Датчики полум'я*. Якщо в приміщенні з'являються ознаки відкритого вогню або тліючого вогнища, датчики активізуються.

✓ *Димові*. Виявляють присутність диму в повітрі.

✓ *Комбіновані*. . Найбільш надійні та точні сповіщувачі. Вони налаштовані на визначення кількох ознак пожежі.

Рис. 1 Пожежні сповіщувачі: а - ручний ИР-П; б - тепловий ИП-105; в - димовий ИПД-1; г - сповіщувач полум'я ИП

Всі види електричної пожежної сигналізації, незалежно від її системи, складаються з трьох основних частин: сповіщувачів, що подають сигнал про пожежу; приймальної станції, призначеної для прийому поданих від сповіщувачів сигналів про пожежу та автоматичної подачі тривоги; *системи проводів*, що з'єднують сповіщувачі з приймальною станцією (рис 2).

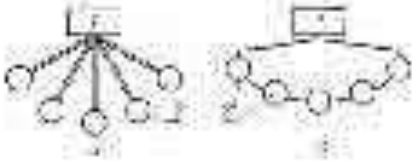

Рис.2 Схема пристрою систем електричної пожежної сигналізації

а - променева; *б* - шлейфному (кільцева). Умовні позначення: *1* - приймальні станції; *2* - пожежні сповіщувачі, з'єднані проводами зі станцією

Системи електричної пожежної сигналізації встановлюються як автоматичного, так і ручного дії. Залежно від способу з'єднання проводами сповіщувачів з приймальною станцією пожежна сигналізація може бути променевої (радіальної) або шлейфному (кільцевий).

При виборі та розміщенні автоматичних пожежних сповіщувачів необхідно керуватися вимогами та рекомендаціями ДБН В.2.5-13-98.

Всі приміщення повинні бути забезпечені первинними засобами пожежогасіння: пожежним водопостачанням (пожежні крани ПК), пожежні щити з набором пожежного інструменту,

Приміщення оснащені вуглекислотними або порошковими вогнегасниками. У випадку виникнення пожежі необхідно відключити електроживлення, викликати по телефону 101 пожежну команду, евакуювати людей із приміщення відповідно до плану евакуації і приступити до ліквідації пожежі.

ВИСНОВКИ

Використання програмного забезпечення для адміністрування надлишкового дискового масиву дозволяє користувачеві стежити за його роботою і віддалено налаштовувати з будь-якого комп'ютера, не вимагаючи при цьому спеціальних знань з керування RAID-масивом. Оповіщення адміністратора при виявлення будь-яких помилок роботи надлишкового дискового масиву дозволить оперативно реагувати на них.

У результаті проведених робіт по розробці алгоритмічного та програмного забезпечення віддаленого керування надлишковим дисковим масивом був реалізований програмний модуль Агент, що зв'язує призначений для користувача інтерфейс з RAID-контролером.

В рамках дипломного проекту були отримані наступні результати:

- була проведена дослідницька робота, на підставі якої була створена структурна схема роботи як забезпечення віддаленого керування надлишковим дисковим масивом в цілому, так і конкретно модуля Агент;
- розроблено схему функціонування програмного забезпечення віддаленого керування надлишковим дисковим масивом;
- були розроблені схеми алгоритмів функціонування програмного забезпечення віддаленого керування надлишковим дисковим масивом, зокрема модулю Агент: алгоритм для пошуку RAID-контролерів, алгоритм опитування RAID-масиву і інші;
- повністю реалізований програмний модуль Агент на мові програмування C++ у складі проекту;
- розроблено модуль для тестування працездатності RAID-контролеру.

Тестування показало, що створене програмне забезпечення задовольняє технічним вимогам та дозволяє системному адміністраторові швидко реагувати при виявленні будь-яких помилок роботи надлишкового дискового масиву, коректувати роботу дискової підсистеми.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Панюкова, Т. А. Языки и методы программирования. Создание простых GUI-приложений с помощью Visual C++. Учебное пособие / Т.А. Панюкова, А.В. Панюков. – М.: Либроком, 2015. – 144 с.
2. Дьюхарст Программирование на C++ / Дьюхарст, Старк Стефан; Кэти. – М.: ДиаСофт, 2015. – 272 с.
3. Понамарев, Вячеслав Программирование на C++/C# в Visual Studio .NET 2003 / Вячеслав Понамарев. – М.: БХВ-Петербург, 2013. – 352 с.
4. Страуструп, Бьерн Дизайн и эволюция C++ / Бьерн Страуструп. – М.: ДМК Пресс, 2016. – 446 с.
5. Фридман, А. C/C++. Архив программ / А. Фридман, Л. Кландер, М. Михэлис, и др.. – М.: ЗАО Издательство БИНОМ, 2016. – 640 с.
6. Халперн Стандартная библиотека C++ / Халперн, Пабло. – М.: Вильямс, 2014. – 336 с.
7. Буч Г. «Объектно-ориентированный анализ и проектирование с примерами приложений на C++». – М.: Бином, 1998.
8. Ирэ Пол. Объектно-ориентированное программирование с использованием C++: Пер. с англ. – Киев: ДиаСофт, 1995.
9. Сэм Канер, Джек Фолкс, Енг Кек Нгуен. «Тестирование программного обеспечения». – Киев: ДиаСофт, 2000.
10. Спецификации RAID-контроллеров. – http://www.accusys.com.tw
11. Д. Дж. Круглински. Программирование на Microsoft Visual C++ 6.0 (пер. с англ.). – СПб.: Питер, 2003.
12. Остроух А.В. Ввод и обработка цифровой информации: учебник для нач. проф. образования / А.В. Остроух. –М.: Издательский центр «Академия», 2012. – 288 с. – ISBN978-5-7695-9457-1.
13. Помазанов А.В., Остроух А.В. Создание и тестирование распределённой системы работы с удалёнными узлами // Автоматизация и современные технологии. – 2014. – №7. – С. 17–23.

ДОДАТОК А. Фрагмент лістингу програмного модулю "Агент" на мові С++

```
//////////////////////////////////
/// Agent-server Main Class
//////////////////////////////////
CAgnt::CAgnt()
{
    m_pSocket=NULL;
}

CAgnt::~CAgnt()
{
    if(m_pSocket)
    {
        m_pSocket->Send(MSG_BREAK_CONNECT);
        Sleep(100);
        m_pSocket->Close();
                delete m_pSocket;
                m_pSocket =NULL;
    }
        if (m_LogFile.m_hFile!=(UINT)INVALID_HANDLE_VALUE)
                m_LogFile.Close();
}

//////////////////////////////////// CAgnt Operations

BOOL CAgnt::Init(LPTSTR pszLogFileName,char* pszHostName,BYTE bAdapter,
BYTE bTargetID,BYTE bLUN)
{
    //BOOL CAgnt::Init(CMainView* pView)
    strcpy(m_szHostName,pszHostName);
    m_bAdapter=bAdapter;
    m_bTargetID=bTargetID;
    m_bLUN=bLUN;
        if(pszLogFileName)m_LogFile.Open(pszLogFileName,CFile::modeWrite|CFile::
shareDenyWrite);
        if(m_LogFile)
        {
                if(m_LogFile.m_hFile!=(UINT)INVALID_HANDLE_VALUE)
m_LogFile.SeekToEnd();
        }
        PHOSTENT phe;
            If ((phe=gethostbyname(m_szHostName))==NULL)return FALSE;
        int count=0;
        CString Temp;
        Temp.Format("Trying connect to %s ...",m_szHostName);
        DisplayMsg(Temp);
        while(phe->h_addr_list[count]!=NULL)
        {
                        //Adress iteration
            Temp.Format("%u.%u.%u.%u",
                                (unsigned char)phe->h_addr_list[count][0],
                                (unsigned char)phe->h_addr_list[count][1],
```

```
                        (unsigned char)phe->h_addr_list[count][2],
                        (unsigned char)phe->h_addr_list[count][3]);
        if(ConnectSocket(Temp,161))
        {
                        //Connection is established
          Temp+=" - found !\n";
          DisplayMsg(Temp);
          DisplayMsg("Connecting to Raid ...");

                        if(!GetConnectA(m_bAdapter,m_bTargetID,m_bLUN))
return FALSE;
          DisplayMsg("Connected !\n");
          return TRUE;
        }//Connection is established
        count++;
    }//Adress iteration
    return FALSE;
}//BOOL CAgnt::Init(CMainView* pView)

BOOL CAgnt::GetConnectA(BYTE bAdapter,BYTE bTargetID,BYTE bLUN)
{//GetConnect(...)
    m_bAdapter=bAdapter;
    m_bTargetID=bTargetID;
    m_bLUN=bLUN;
    if(!m_pSocket->Send(MSG_GET_CONNECT))return FALSE;
    if(!m_pSocket->Read())m_dwError|= ERR_READING_SOCKET;
    return !(BOOL)m_dwError;
}//GetConnect(...)

BOOL CAgnt::BreakConnectA(void)
{//BreakConnect(...)
     if(!m_pSocket->Send(MSG_BREAK_CONNECT))return FALSE;
          delete m_pSocket;
          m_pSocket = NULL;
//   if(!m_pSocket->Read())m_dwError|= ERR_READING_SOCKET;
     return !(BOOL)m_dwError;
}//BreakConnect(...)

BOOL CAgnt::GetParamA(LPSTR pszSection,LPSTR pszVariable,LPSTR pszValue)
{//GetParam(...)
    m_pszSection=pszSection;
    m_pszVariable=pszVariable;
    if(!m_pSocket->Send(MSG_GET_PARAM))return FALSE;
    if(!m_pSocket->Read())m_dwError|= ERR_READING_SOCKET;
    strcpy(pszValue,m_szValue);
    return !(BOOL)m_dwError;
}//GetParam(...)

BOOL CAgnt::SetParamA(LPSTR pszSection,LPSTR pszVariable,LPSTR pszValue)
{//SetParam(...)
    m_pszSection=pszSection;
    m_pszVariable=pszVariable;
    memset(m_szValue,0,sizeof(m_szValue));
    strcpy(m_szValue,pszValue);
```

```
    if(!m_pSocket->Send(MSG_SET_PARAM))return FALSE;
    if(!m_pSocket->Read())m_dwError|=ERR_READING_SOCKET;
    return !(BOOL)m_dwError;
}//SetParam(...)

BOOL CAgnt::GetRaidInfoA(CRaidInfo* pRi)
{//GetRaidInfo(...)
    char buf[1024];
    memset(m_RaidInfo.m_szProductionID,0,sizeof(m_RaidInfo.m_szProductionID));
    memset(m_RaidInfo.m_szModelName,0,sizeof(m_RaidInfo.m_szModelName));
    memset(m_RaidInfo.m_szSerialNumber,0,sizeof(m_RaidInfo.m_szSerialNumber));
    memset(m_RaidInfo.m_szFirmware\VerionNO,0,
sizeof(m_RaidInfo.m_szFirmware\VerionNO));

//          ExecCommand(NULL,"getpage 0;",&buf);


    int i;
  if(!m_pSocket->Read())m_dwError|=ERR_READING_SOCKET;
  strcpy(pRi->m_szProductionID,m_RaidInfo.m_szProductionID);
  strcpy(pRi->m_szModelName,m_RaidInfo.m_szModelName);
  strcpy(pRi->m_szSerialNumber,m_RaidInfo.m_szSerialNumber);
  strcpy(pRi->m_szFirmware\VerionNO,m_RaidInfo.m_szFirmware\VerionNO);
  pRi->m_dwStripeSize=m_RaidInfo.m_dwStripeSize;
  pRi->m_b\WriteBufferEnable=m_RaidInfo.m_b\WriteBufferEnable;
  pRi->m_bHostChannelNumber=m_RaidInfo.m_bHostChannelNumber;
  pRi->m_bDiskChannelNumber=m_RaidInfo.m_bDiskChannelNumber;
  pRi->m_bSlotNumber=m_RaidInfo.m_bSlotNumber;
  pRi->m_dwRamSize=m_RaidInfo.m_dwRamSize;
  pRi->m_bHitRatio=m_RaidInfo.m_bHitRatio;
  pRi->m_bIsReady=m_RaidInfo.m_bIsReady;
  for(i=0;i<8;i++)pRi->m_bDisk\Width[i]=m_RaidInfo.m_bDisk\Width[i]; //TRUE-Enable
  for(i=0;i<8;i++)pRi->m_bDiskSpeed[i]=m_RaidInfo.m_bDiskSpeed[i];
// 0-fast2timing 1-ultraTiming 2-Ultra2timing
  for(i=0;i<2;i++)pRi->m_bHostID[i]=m_RaidInfo.m_bHostID[i];
//0xFF - multiple ID
  for(i=0;i<2;i++)pRi->m_bHostQueuing[i]=m_RaidInfo.m_bHostQueuing[i];
  for(i=0;i<2;i++)pRi->m_bHostSpeed[i]=m_RaidInfo.m_bHostSpeed[i];
// 0-fast2timing 1-ultraTiming 2-Ultra2timing
  for(i=0;i<2;i++)pRi->m_bHost\Width[i]=m_RaidInfo.m_bHost\Width[i];// 8 <=> 16
  for(i=0;i<2;i++)pRi->m_bHostTerminator[i]=m_RaidInfo.m_bHostTerminator[i];
//TRUE-Enable
  for(i=0;i<2;i++)pRi->m_bHostMaintenance[i]=m_RaidInfo.m_bHostMaintenance[i];
// 0-Single Ended 1-LVD 2-HVD
  for(i=0;i<2;i++){for(int j=0;j<8;j++)
pRi->m_HostLUNmap[i].bSliceID[j]=m_RaidInfo.m_HostLUNmap[i].bSliceID[j];}
  for(i=0;i<2;i++){for(int j=0;j<8;j++)
pRi->m_HostLUNmap[i].bArrayID[j]=m_RaidInfo.m_HostLUNmap[i].bArrayID[j];}
  for(i=0;i<96;i++)
  {
    pRi->m_SlotMap[i].bScsiID=m_RaidInfo.m_SlotMap[i].bScsiID;
    pRi->m_SlotMap[i].bDiskChanID=m_RaidInfo.m_SlotMap[i].bDiskChanID;
  }
  return !(BOOL)m_dwError;
```

```
}//GetRaidInfo(...)
BOOL CAgnt::GetRaidArrayInfoA(int n4RaidArray,CRaidArrayInfo* pRai)
{//GetRaidArrayInfo(...)
	m_bPage=1+n4RaidArray;
	if(!m_pSocket->Send(MSG_GET_STATUS))return FALSE;
	if(!m_pSocket->Read())m_dwError|=ERR_READING_SOCKET;
	int i;
	for(i=0;i<4;i++)
	{//duplicate structures
		pRai[i].bInit=m_RaidArrayInfo[i].bInit;//(0...100)% 0xFF=IsDone
		pRai[i].bAdd=m_RaidArrayInfo[i].bAdd;//(0...100)% 0xFF=IsDone
		pRai[i].bCheck=m_RaidArrayInfo[i].bCheck;//(0...100)% 0xFF=IsDone
		pRai[i].bExpand=m_RaidArrayInfo[i].bExpand;// (0...100)%
OtherWise=NoExpansionGoing
		pRai[i].bStatus=m_RaidArrayInfo[i].bStatus;//Is anybitwise combination of
WARN_UPS | WARN_TEMPERATURE | WARN_FAN | WARN_POWER
		pRai[i].bDiskIsChanged=m_RaidArrayInfo[i].bDiskIsChanged;
// It sets when some disk has changed from previous this page
		pRai[i].bArrayIsChecked=m_RaidArrayInfo[i].bArrayIsChecked;
// It sets when array checking doing
		pRai[i].bStripeSize=m_RaidArrayInfo[i].bStripeSize;
		pRai[i].bRaidLevel=m_RaidArrayInfo[i].bRaidLevel;
//{0,1,3,4,5,6} note: value 6 indicate level <0+1>
		pRai[i].bDisksQuantity=m_RaidArrayInfo[i].bDisksQuantity;
		pRai[i].dwCapacity=m_RaidArrayInfo[i].dwCapacity;
// N sectors, that RAID has
		int j;
		for(j=0;j<8;j++)pRai[i].dwSize[j]=m_RaidArrayInfo[i].dwSize[j];
//The size in mega bytes that the j-th slice, that array has
		for(j=0;j<32;j++)pRai[i].bMembersID[j]=m_RaidArrayInfo[i].bMembersID[j];
//The j-th byte store the slot ID of the j-th member
		for(j=0;j<128;j++)pRai[i].bSlotStatus[j]=m_RaidArrayInfo[i].bSlotStatus[j];
//Is anybitwise combination of ST_ONLINE | ST_BELONG | ST_REBILD
	}//duplicate structures
	return !(BOOL)m_dwError;
}//GetRaidArrayInfo(...)

BOOL CAgnt::GetSlotArrayInfoA(int n16Slot,CSlotArrayInfo* pSai)
{//GetSlotArrayInfo(...)
	m_bPage=16+n16Slot;
	if(!m_pSocket->Send(MSG_GET_STATUS))return FALSE;
	if(!m_pSocket->Read())m_dwError|=ERR_READING_SOCKET;
	for(int i=0;i<16;i++)
	{//duplicate structures
		strcpy(pSai[i].m_szDiskVendorID,m_SlotArrayInfo[i].m_szDiskVendorID);
		pSai[i].wBadBlock=m_SlotArrayInfo[i].wBadBlock;
// N BadBlocks of disk media
		pSai[i].dwSize=m_SlotArrayInfo[i].dwSize;//Disk Size
	}//duplicate structures
	return !(BOOL)m_dwError;
}//GetSlotArrayInfo(...)

BOOL CAgnt::ResetA(void)
{//Reset(...)
```

```
    if(!m_pSocket->Send(MSG_RESET))return FALSE;
    if(!m_pSocket->Read())m_dwError|=ERR_READING_SOCKET;
    return !(BOOL)m_dwError;
}//Reset(...)

BOOL CAgnt::ShutDownA(void)
{//ShutDown(...)
    if(!m_pSocket->Send(MSG_SHUTDOWN))return FALSE;
    if(!m_pSocket->Read())m_dwError|=ERR_READING_SOCKET;
    return !(BOOL)m_dwError;
}//ShutDown(...)

BOOL CAgnt::AddSlotA(BYTE bSlotNumber)
{//AddSlot(...)
    m_bSlotNumber=bSlotNumber;
    if(!m_pSocket->Send(MSG_ADD_SLOT))return FALSE;
    if(!m_pSocket->Read())m_dwError|=ERR_READING_SOCKET;
    return !(BOOL)m_dwError;
}//AddSlot(...)

BOOL CAgnt::RemoveSlotA(BYTE bSlotNumber)
{//RemoveSlot(...)
    m_bSlotNumber=bSlotNumber;
    if(!m_pSocket->Send(MSG_REMOVE_SLOT))return FALSE;
    if(!m_pSocket->Read())m_dwError|=ERR_READING_SOCKET;
    return !(BOOL)m_dwError;
}//RemoveSlot(...)

DWORD CAgnt::GetLastErrorA(LPSTR pszMessageText)
{//GetLastError(...)
    pszMessageText=&m_szMessageText[0];
    return m_dwError;
}//GetLastError(...)

BOOL CAgnt::ConnectSocket(LPCTSTR lpszAddress, UINT nPort)
{
    m_pSocket=new CAgntSocket(this);
    if(!m_pSocket->Create())
    {
        CString Temp;
        Temp.Format("Windows sockets initialization failed.\n");
        DisplayMsg(Temp);
        delete m_pSocket;
        m_pSocket = NULL;
        return FALSE;
    }
    m_pSocket->Init();
    if(!m_pSocket->Connect(lpszAddress,nPort))
    {
        CString Temp;
        Temp.Format("Can't connect to %s\n",lpszAddress);
        DisplayMsg(Temp);
        delete m_pSocket;
        m_pSocket=NULL;
```

```
            return FALSE;
        }
        return TRUE;
}

void CAgnt::DisplayMsg(LPCTSTR lpszText)
{
    if(m_LogFile)
    {
            if(m_LogFile.m_hFile!=(UINT)INVALID_HANDLE_VALUE)
        m_LogFile.Write(lpszText,strlen(lpszText));
    }
}

void CAgnt::Assemble(BYTE What)
{
        int i;
        WORD Len;
        m_dwError=0;
        memset(Data,0,sizeof(Data));
        Data[0]=What;
        switch(What)
        {
            case MSG_GET_CONNECT:
            case MSG_BREAK_CONNECT:
                Len=9;
                Data[1]=HIBYTE(Len);//Hi Lenght
                Data[2]=LOBYTE(Len);//Lo Lenght
                Data[3]=HTCQ;
                Data[4]=m_bAdapter;
                Data[5]=m_bTargetID;
                Data[6]=m_bLUN;
                Data[7]=(BYTE)(CRC(Len)>>8);
                Data[8]=(BYTE)CRC(Len);
               break;
            case MSG_GET_STATUS:
                Len=10;
                Data[1]=HIBYTE(Len);//Hi Lenght
                Data[2]=LOBYTE(Len);//Lo Lenght
                Data[3]=HTCQ;
                Data[4]=0x5F;//Command Signature
                Data[5]=0;//m_bLUN<<5;//LUN*32
                Data[6]=0x01;//Command Code
                Data[7]=m_bPage;//Page Number
                Data[8]=(BYTE)(CRC(Len)>>8);
                Data[9]=(BYTE)CRC(Len);
               break;

                case MSG_COMMAND:
                        {CAgntCommand c;
                          c << (DWORD)mp_Host <<mp_CommandString;
                              c.FillData(*this);
                        }break;
        }
```

```
}
BYTE CAgnt::DesAssemble()
{
	int i,j;
	BYTE Command=Data[0];
	switch(Command)
	{
		case MSG_RET_STATUS:

m_dwError=(((DWORD)Data[4])<<24)|(((DWORD)Data[5])<<16)|(((DWORD)Data[6])<<8)
|((DWORD)Data[7]);
			if(Data[8]!=m_bPage)m_dwError|=ERR_WRONG_PAGE;
			for(i=0;i<0x200;i++)Data[i]=Data[i+9];//5 Bytes Shift

		case MSG_RET_PARAM:
		  memset(m_szValue,0,sizeof(m_szValue));
		  for(i=0;i<STRING_LENGTH-1;i++)m_szValue[i]=Data[i+4];//RetVal
		  break;
		case MSG_RESPONSE:

			m_dwError|=(((DWORD)Data[4])<<24)|(((DWORD)Data[5])<<16)
|(((DWORD)Data[6])<<8)|((DWORD)Data[7]);

			memset(m_szMessageText,0,sizeof(m_szMessageText));
		for(i=0;i<STRING_LENGTH-1;i++)m_szMessageText[i]=Data[i+8];//TextDescription
			break;
				case MSG_CMD_RESPONSE:

		break;

	}
	return Command;
}

BYTE CAgnt::HTC(void)
{
	int i;
	BYTE htc=0;
	for(i=0;i<3;i++)
	{
		htc+=Data[i];
	}
	return -htc;
}

WORD CAgnt::CRC(WORD len)
{
	WORD i,crc=0;
	BYTE shift;
	for(i=0;i<len-2;i++)
	{
		shift=8;
		crc=crc^(Data[i]<<8);
		while(shift)
```

```
            {
                crc=crc<<1;
                if(crc&0x8000)crc=crc^0x1021;
                shift--;
            }
        }
        return crc;
}

void CAgnt::Serialize(CArchive& ar)
{
        WORD i;
        m_dwError=0;
        if(ar.IsStoring())
        {
            ar<<(WORD)MSG_MARKER;
            for(i=0;i<4;i++)ar<<Data[i];
            DataLen=256*Data[1]+Data[2];
            for(i=4;i<DataLen;i++)ar<<Data[i];
            ar<<(WORD)MSG_MARKER;
        }
        else
        {
            BYTE b;
            WORD w;
            memset(Data,0,sizeof(Data));
            ar>>w;
            if(w!=MSG_MARKER){m_dwError=ERR_INVALID_MARKER;goto end;}
            for(i=0;i<4;i++)ar>>Data[i];
            if(Data[3]!=HTC()){m_dwError=ERR_BROKEN_HEADER;goto end;}
            DataLen=256*Data[1]+Data[2];
            if(DataLen>1024){m_dwError=ERR_INVALID_LENGTH;goto end;}
            for(i=4;i<DataLen;i++)
            {
                if(ar.IsBufferEmpty()){m_dwError=ERR_DATA_LOOSED;goto end;}
                ar>>Data[i];
            }
            if(CRC(DataLen)!=256*Data[DataLen-2]+
Data[DataLen-1]){m_dwError=ERR_BAD_CRC;goto end;}

end:while(!ar.IsBufferEmpty())ar>>b;
            return;
        }
}
#ifdef _DEBUG
void CAgnt::AssertValid()const{CObject::AssertValid();}
void CAgnt::Dump(CDumpContext& dc)const{CObject::Dump(dc);}
#endif //_DEBUG

#define __MSG_SEND(MSG) \
  if(!m_pSocket->Send(MSG))return FALSE;\
  if(!m_pSocket->Read())m_dwError=ERR_READING_SOCKET
```

```
//////////////////////////////////
// CAgntSocket - open new socket for manager
//////////////////////////////////

IMPLEMENT_DYNAMIC(CAgntSocket, CSocket)

CAgntSocket::CAgntSocket(CAgnt* pAgnt)
{
	m_pAgnt=pAgnt;
	m_pFile=NULL;
	m_pArchiveIn=NULL;
	m_pArchiveOut=NULL;
}

void CAgntSocket::Init(void)
{
	m_pFile=new CSocketFile(this);
	m_pArchiveIn=new CArchive(m_pFile, CArchive::load);
	m_pArchiveOut=new CArchive(m_pFile, CArchive::store);
}

CAgntSocket::~CAgntSocket()
{
	if(m_pArchiveOut)
	 {
		m_pArchiveOut->Abort();
		delete m_pArchiveOut;
		m_pArchiveOut = NULL;

	 }
	if(m_pArchiveIn)
	 {
		m_pArchiveIn->Abort();
		delete m_pArchiveIn;
	m_pArchiveIn = NULL;

	 }

	if(m_pFile) {delete m_pFile;m_pFile=NULL;}

}

void CAgntSocket::OnReceive(int nErrorCode)
 {//CAgntSocket::OnReceive(int nErrorCode)
	CSocket::OnReceive(nErrorCode);
//	Read();
 }//CAgntSocket::OnReceive(int nErrorCode)

BOOL CAgntSocket::Read(void)
{
	BOOL Res;
	CString strTemp;
	TRY
```

```
	{
		m_pAgnt->Serialize(*m_pArchiveIn);
		if(m_pAgnt->m_dwError)
		{
			strTemp.Format(" - error : 0x%08X",m_pAgnt->m_dwError);
			Res=FALSE;
		}
		else
		{
			strTemp.Format(" - response : 0x%02X",m_pAgnt->DesAssemble());
			Res=TRUE;
		}
		m_pAgnt->DisplayMsg(strTemp);
	}
	CATCH(CFileException, e)
	{
		m_pArchiveIn->Abort();
		delete m_pArchiveIn;
		m_pArchiveIn=new CArchive(m_pFile,CArchive::load);
		strTemp.Format("Server has reset the connection");
		m_pAgnt->DisplayMsg(strTemp);
		Res=FALSE;
	}
	END_CATCH
	return Res;
}

BOOL CAgntSocket::Send(BYTE Command)
{
	BOOL Res;
	CString strTemp;
	m_pAgnt->Assemble(Command);
	strTemp.Format("Command 0x%02X :",Command);
	m_pAgnt->DisplayMsg(strTemp);
	TRY
	{
		m_pAgnt->Serialize(*m_pArchiveOut);
		m_pArchiveOut->Flush();
		Res=TRUE;
	}
	CATCH(CFileException,e)
	{
		m_pArchiveOut->Abort();
		delete m_pArchiveOut;
		m_pArchiveOut=new CArchive(m_pFile,CArchive::store);
		strTemp.Format("Server has reset the connection");
		m_pAgnt->DisplayMsg(strTemp);
		Res=FALSE;
	}
	END_CATCH
	return Res;
}
```