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

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS”

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

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Петренюк В.І.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 122 *"Комп'ютерні науки"*

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

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

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

Кузю Олександру Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS*

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

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

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

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS*

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

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

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

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В.	05.10.2023	14.11.2023
Охорона праці	Оришака О.В.	06.10.2023	16.11.2023

7. Дата видачі завдання « 6 » вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2023 р.	
3.	Розробка моделі компонента	20.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2023 р.	
6.	Програмування алгоритмів	10.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2023 р.	
9.	Оформлення ПЗ	17.11.2023 р.	
10.	Попередній захист роботи	10.12.2023 р.	

Дата видачі завдання
« 6 » вересня 2023 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2023 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Кузь О.В. Дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи навігації та моніторингу файлової системи iOS.

Метою розробки є дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS.

Об'єктом дослідження є процес навігації та моніторингу файлової системи iOS.

Предметом дослідження є методи навігації та моніторингу файлової системи iOS.

Методи дослідження базуються на методах побудови файлових систем, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи навігації та моніторингу файлової системи iOS.

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

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

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

Програму розроблено в середовищі Objective-C.

Ключові слова: комп'ютерні науки, файлова система, iOS

ABSTRACT

Kuz O.V. Research and software implementation of the iOS file system navigation and monitoring system. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for the system of navigation and monitoring of the iOS file system.

The purpose of the development is the research and software implementation of the iOS file system navigation and monitoring system.

The object of research is the process of navigation and monitoring of the iOS file system.

The subject of research is methods of navigation and monitoring of the iOS file system.

Research methods are based on methods of building file systems, methods of mathematical statistics, and methods of software development.

The result of the work is a software implementation of the iOS file system navigation and monitoring system.

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

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

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

The program was developed in the Objective-C environment.

Keywords: computer science, file system, iOS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

Apple iOS	–	операційна система Apple
Darwin	–	ядро операційної системи Apple iOS
iPad	–	планшетні комп'ютери
iPhone	–	смартфон
iPod Touch	–	медіаплеєр
Mac OS	–	операційна система Apple

ВСТУП

Актуальність теми. Історія обміну файлів між комп'ютером і телефоном не нова. На наше століття довелося багато всіляких програм. Громіздкі й гальмові пакети, що містять тонни помилок, перетаскування пісень папками на SD-картку, спеціальні кабелі й не менш спеціальні драйвери до них, у порівнянні із цим, зараз, у другому десятилітті XXI століття, усілякі операції відновлення виробляються тривіально. З 2011 року iOS-пристрою взагалі можна не підключати до комп'ютера, або синхронізуючи їх по Wi-Fi, або користуючись винятково iCloud. Однак потреба у файлових менеджерах для iOS-пристроїв поки що не відпала остаточно, особливо це стосується тих з них, на яких зроблений джейлбрейк. На сьогоднішній день у магазині App Store, є величезна кількість додатків, які асоціюються із клієнтами для соціальних мереж за типом twitter або vk, є файлові менеджери, різні клієнти, що працюють із відеороликами й фотографіями на інших сервісах. Однією фразою можна сказати, що додатків досить. Однак, далеко не всі з них можуть заслужити належної оцінки від користувача. Десь інтерфейс не дуже гарний, десь багато багів, малий функціонал та інші тому подібні проблеми. У прихильників Android завжди є в арсеналі головний аргумент проти iOS-користувачів – довід про закритість платформи. Мол, не можна файли по папках розкидати, не можна без джейлбрейку файлову систему подивитися. Так, папки /var, /root і їм подібні – не можна, однак це не заважає існуванню файлових менеджерів в App Store. Наприклад, після установки iFiles ви одержуєте практично повну файлову систему на iOS-пристрої без усяких джейлбрейків, нехай і в рамках одного додатка.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS.

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

– Огляд існуючих систем навігації та моніторингу файлової системи iOS.

– Дослідження системи навігації та моніторингу файлової системи iOS.

– Програмна реалізація системи навігації та моніторингу файлової системи iOS.

Об'єктом дослідження є процес навігації та моніторингу файлової системи iOS.

Предметом дослідження є методи навігації та моніторингу файлової системи iOS.

Методи дослідження базуються на методах побудови файлових систем, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод навігації та моніторингу файлової системи iOS.

– Розроблено вітчизняний продукт навігації та моніторингу файлової системи iOS, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі навігації та моніторингу файлової системи iOS.

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

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

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для реалізації навігації та моніторингу файлової системи iOS. Єдиний офіційний програмний продукт, заявлений як файловий менеджер – це iTunes. Його завдання – завантажувати в iPhone звукові файли, зображення, відео й додатка, а також вивантажувати контент із iPhone на диск комп'ютера.

Незважаючи на консервативну політику iOS, існують легальні способи накачувати в телефон «несумісний» контент і переглядати його. Один з них – це покупка спеціальних додатків в AppStore.

Завдання виглядає куди складнішим, коли вам потрібно добратися до системних файлів, щось прописати або видалити, або навіть поміняти права на яку-небудь папку, що належить операційній системі. У цьому допоможе розроблювальне, у даному магістерському проекті, програмне забезпечення. Додаток дає повний доступ до закритих файлів і скачати його можуть тільки власники телефонів із джейлбрейком.

1.2 Область застосування

Областю застосування є файлова система iOS. Під час презентації нової операційної системи підкреслили 10 основних змін:

– Пункт керування – подібно Центру повідомлень, Пункт керування (Control Center) доступний за допомогою жесту нагору в нижній частині екрана й забезпечує доступ до таких параметрів як авіарежим, яскравість, керування мультимедіа, AirPlay, AirDrop і швидкий доступ до деяких додатків: ліхтарик, компас, калькулятор і камера.

– Багатозадачність – iOS заснована на витісняючій багатозадачності, уведеної в iOS. Підтримується фонове відновлення додатків.

– Safari – в Safari стало доступно поле «розумного» пошуку, уперше представлене в Safari для OS X Mountain Lion. Інші зміни містять у собі нескінченне число вкладок, батьківський контроль, поліпшені обмін посиланнями в Twitter і Список для читання.

– AirDrop – за допомогою функції AirDrop (уперше представлена в OS X Lion), доступної для iPhone, iPad, iPad mini, і iPod touch, можна обмінюватися файлами (фотографіями, відео, картками контактів і т.д.) з іншими користувачами iOS-пристроїв.

– Камера – новий інтерфейс камери дозволяє переходити між чотирма різними режимами: відео, фото, квадратні й панорамні фото. Також з'явилася можливість накласти на знімки один з дев'яти різних фільтрів.

– Фотографії – додаток «Фото» в iOS використовує інформацію кожної фотографії, щоб сортувати їх за датою, місцем й роком зйомки, а також підтримує завантаження відео у фотопотік.

– Siri – інтерфейс Siri також піддався редизайну. Для кожного представленої мови з'явилася опція вибору між чоловічим і жіночим голосом. Поліпшено інтеграцію із Твіттером, Вікіпедією і пошуковою системою Bing. З'явилася можливість управляти системними налаштуваннями за допомогою голосових команд.

– iOS у Машині – «iOS у Машині» використовує Siri, інтегровану в окремі моделі автомобілів для керування навігацією, телефоном, музикою й iMessage на екрані автомобіля.

– App Store – нові функції: пошук по віковій групі, «Поруч із мною», а також реалізоване автоматичне відновлення додатків.

– Музика й iTunes Radio – поряд зі змінами інтерфейсу, у додаток Музика тепер убудована iTunes Radio (сервіс потокового відтворення музики, на зразок Spotify).

Інші зміни містять у собі FaceTime Audio, багатосторінкові папки, чорний список контактів, а також нові можливості Find My iPhone, що перешкоджають використанню загубленого або украденого iPhone або iPad сторонніми.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Проведемо огляд існуючих файлових менеджерів для iPhone.

Doctape: файловий менеджер для iPhone

Doctape – це унікальна у своєму роді програма, що просто покликана задовольняти її власника, вона дійсно може виконати потреби майже всіх користувачів яблучних пристроїв. Дана розробка може відтворювати більше 80 форматів, починаючи від стандартних документів пакета Microsoft Office до аудіозаписів і великих HD-відеофайлів.

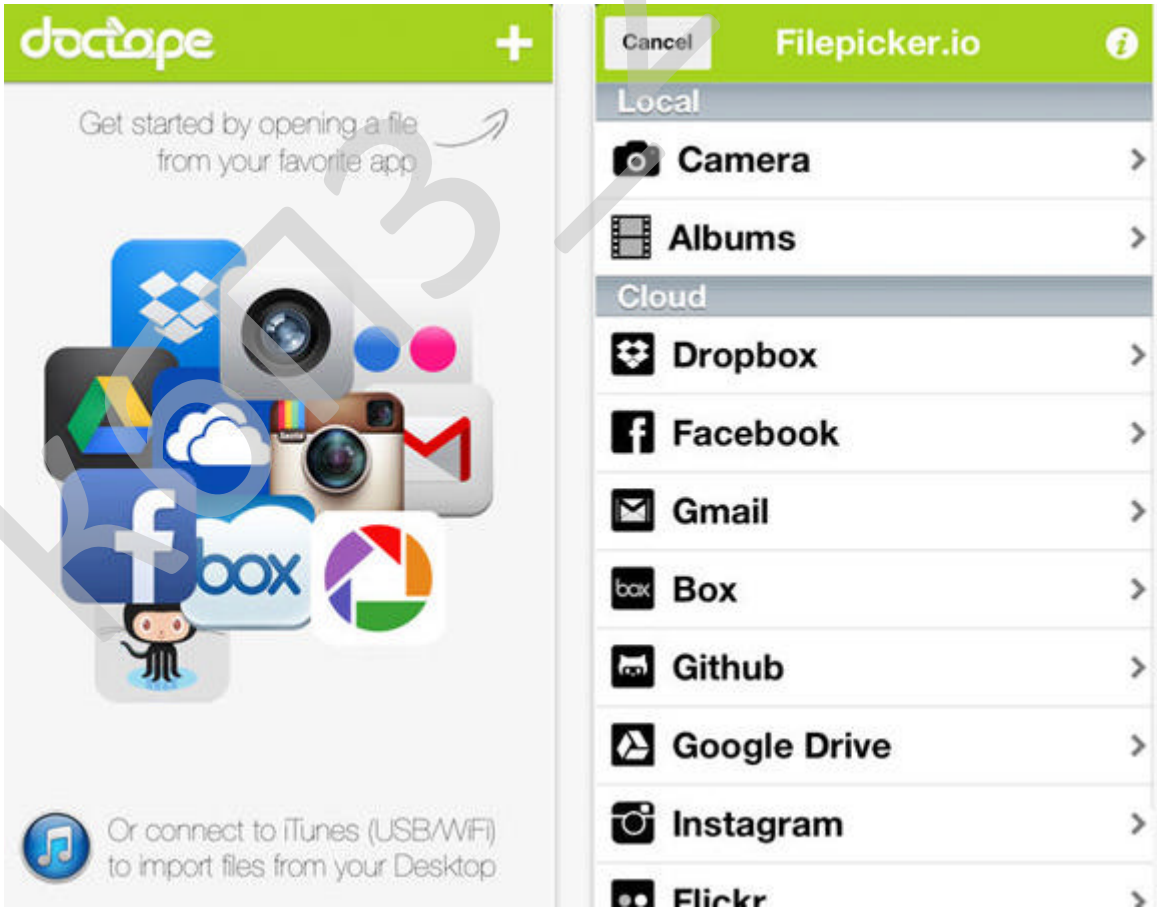

Рисунок 2.1 – Інтерфейс користувача Doctape

Інтерфейс програми досить таки простий, він має всього дві кнопки зверху, пошуковий рядок з підбором відповідей і розподілом. Всі документи представляються користувачеві у вигляді простого списку із вказівкою файлу, його ваги й формату. Кожний файл має якесь прев'ю

Крім цього, Doctape має ряд додаткових функцій. По-перше, це автоматичне збереження всіх завантажених файлів у хмару. Це забезпечить користувача безпекою даних, а так само дасть можливість швидко публікувати резервовані файли. Надати посилання на будь-який файл можна в одне торкання, а так само реалізована публікація в соціальні мережі.

Підтримувані формати:

– Всі документи з пакета Microsoft Word (doc/docx, xls/xlsx, ppt/pptm/pptx).

– Open Office (odt, ods, sxw, stw).

– Основні текстові файли (rtf, txt, csv).

– Відео дані (mp4, mkv, mov, avi, ogv, divx, wmv, asf, m4v, webm, mpg, 3gp, 3gs).

– Аудіо файли (mp3, m4a, ogg, aif, flac, ape, aac, wav, wma).

– Графічні файли (jpg, png, tif, bmp, ai, nef, arf).

– Файл проекту Photoshop (.psd).

– PDF дані.

iExplorer

Розглянемо одну із програм, що дозволяє дуже навіть просто працювати з файлами на і-пристрої – це відомий файловий менеджер iExplorer (раніше він називався iPhone Explorer), що випускається для OS X і Windows.

Хочеться відзначити відразу, що софт залишив досить приємне враження, але почати розмова необхідно із ціни. Раніше програма була повністю безкоштовною, але потім розроблювачі стали використовувати ту ж бізнес-модель, що й конкуренти типу DiskAid. Безкоштовна версія не обмежується за часом використання й числу запусків, вся справа лише в підтримуваних

функціях. Функціональність платної полягає в розширеній роботі з контентом iPhone, iPod touch або iPad.

У цілому виглядає програма приємно. Розроблювачі потрудилися, щоб зробити її максимально схожою на інші додатки, вони зробили так, щоб продукт бачився як частина Маку, і це в них, скажемо чесно, вийшло.

Основні групи файлів ліворуч розбиті по групах. У центрі ми бачимо звичний нам Finder. Зверху розташувалися прості й зрозумілі кнопки з діями, праворуч – сайдбар з інформацією про обраний об'єкт. Як і в Finder, тут підтримуються різні режими перегляду об'єктів – від мініатюр до Cover Flow.

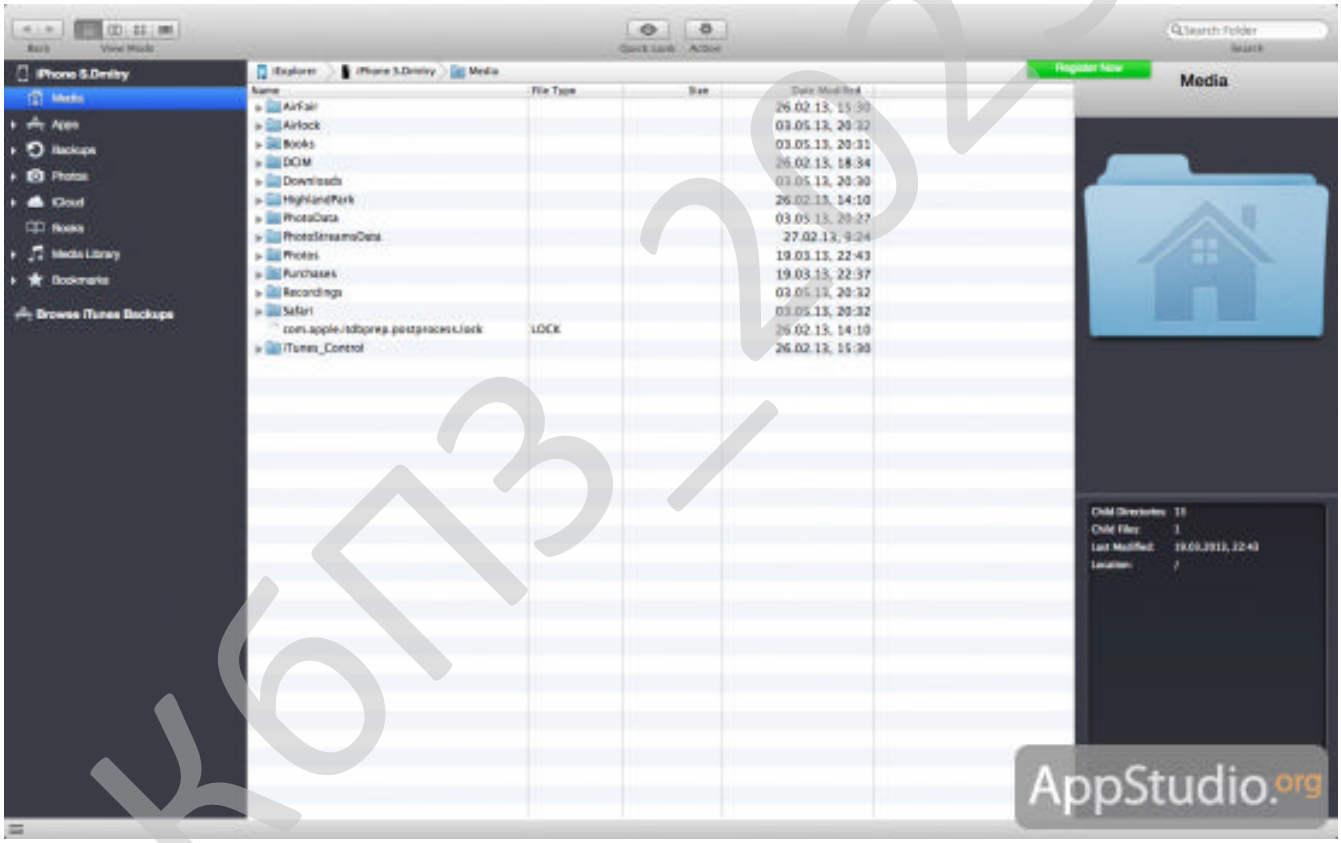

Рисунок 2.2 – Інтерфейс користувача iExplorer

По частині файлових операцій теж усе в порядку, підтримується навіть Quick Look і така цікава функція, як монтування папки у вигляді зовнішнього диска в Finder.

Тим, хто звик до Finder, працювати з iExplorer буде досить зручно. Згруповано дії грамотно, немає великого бажання все переробити, як це іноді буває. Робота з файлами достатня проста, у ній завжди можна досить просто розібратися.

Прийшов час розповісти про більше спеціалізовані інструменти, наприклад, роботі з медіатекою.

Ми активували відповідний пункт ліворуч. Тепер центральний елемент вікна змінився так, щоб нам було зручніше працювати з певним типом інформації, наприклад піснями. Ми можемо просто й наочно переглядати інформацію про їх, ми можемо виконувати з ними стандартні операції, прямо так, начебто б це і є файли, самі звичайні.

Подібним чином улаштована робота майже скрізь у програмі. Вона побудована таким чином, щоб не бути іграшкою для гиків, вона – для рядового користувача.

Для приклада ми відкрили папку Numbers рідного табличного редактора для iOS. Нагадаємо, що працювати з файлами додатків для iOS через iExplorer і подібні йому файлові менеджери можна без джейлбрейку. Але при наявності джейла можливості в рази розширюються: так, ви зможете бачити кореневу директорію файлової системи гаджета й одержите доступ на запис до будь-якого об'єкта.

В iExplorer скрупульозно продумана робота з різною користувальницькою інформацією. Для кожного типу даних, будь те історія повідомлень, база контактів, замітки, напам’ятовувачі й події календаря iOS, ви побачите окремий інтерфейс із відповідними спеціалізованими діями й опціями.

Підбиваючи підсумок, скажемо, що програма iExplorer залишила приємне враження. Вона проста в освоєнні, зручна у використанні, відповідає всім канонам Apple. Навіть у безкоштовному виді iExplorer украй корисний. Якщо необхідна, наприклад, робота з повідомленнями, то прийдеться розщедритися на платний варіант, у якому даний функціонал реалізований. Повна версія коштує

$34,99, що вкрай багато. Ми б із задоволенням назвали б iExplorer однією із кращих програм жанру з погляду ергономіки, функціональності і якості. Це було б дійсно так, якби не існував безкоштовний iFunbox, що випускається як під Windows, так і під Мак, і незабаром по можливостях в OS X наздожене як Windows-версію, так і конкурентів.

iFiles

Уперше заглянувши в iFiles, ми побачимо заготовлені для нас папки зі зрозумілими назвами й довідковий файл, що розповість, як управлятися із програмою.

Рисунок 2.3 – Інтерфейс користувача iFiles

Насамперед, давайте пройдемося по налаштуваннях. По-перше, ряд опцій стосуються відображення файлів. Можна показувати або приховувати розширення, показувати прев'ю даних, коли це можливо.

Далі користувач вибирає різні налаштування перегляду тих або інших видів інформації. Для текстових файлів це шрифт, його колір, розмір і так далі; для картинок – опції слайд-шоу. Також можна настроїти якість створюваних заміток і ступінь стиску графіки й відео при імпорті.

Однією із самих цікавих можливостей варто назвати підключення мережних сховищ. Тут перераховані практично всі популярні хмарні сервіси, крім того, можна підключитися до будь-якого FTP-, SSH- або WebDAV-серверу.

Завантажувати тільки з комп'ютера або хмари вам не прийдеться. Усередині iFiles є браузер. Він простій, але зі своїми завданнями справляється.

Про всі підключені або локальні простори можна одержати деяку інформацію, зокрема, вільну й загальну кількість місця для зберігання. Також можна включити синхронізацію з iTunes і ввести пароль на вхід у меню Advanced.

Передача інформації може здійснюватися й по Wi-Fi, але для цього потрібно щоб комп'ютер і iOS пристрій перебували в одній мережі. Як можна догадатися, процес організований за допомогою звичайного браузера.

На жаль, великих переглядачів або редакторів у додаток iFiles не убудовано, тому відео в сторонніх кодеках або інші “нерідні” формати для мобільної платформи Apple практично ніколи не відкриваються.

Але із традиційними форматами документів справа є набагато кращою. Програма відмінно справляється з файлами Microsoft Office, iWork, переглядом RTF і PDF. Однак повторимося: головне в iFiles – все-таки можливості програми як файлового менеджера.

Зате тут є зручні меню сортування по декількох параметрах, кнопки друку, відправлення по електронній пошті або відкриття файлу в іншій програмі. Досить зручно виконаний імпорт і створення нових елементів.

Хоч із iFiles ми й не одержуємо доступ до кореневих папок iOS-пристроїв, зате можемо досить зручно організувати свої дані по тематичних папках. Підтримку величезної кількості хмарних сервісів теж варто записати в актив програми – це єдиний більш-менш прийнятний спосіб поліпшити далеку від ідеалу концепцію роботи з одними файлами в різних програмах, прийняту в iOS. Правда, здобуваючи додаток, не варто забувати про те, що iFiles більшою мірою файловий менеджер, ніж переглядач. У частині всеїдності форматів в App Store у нього багато сильних конкурентів. Але як засіб організації файлів ця програма, безсумнівно, одна із самих гідних в App Store.

PhoneView

Люди роблять Jailbreak, щоб одержати повний доступ до файлової системи iPhone. Однак видаляти, копіювати, перейменовувати, редагувати й т.д. файли на iPhone можна й без джейлбрейку. Для цього вам знадобиться додаток PhoneView.

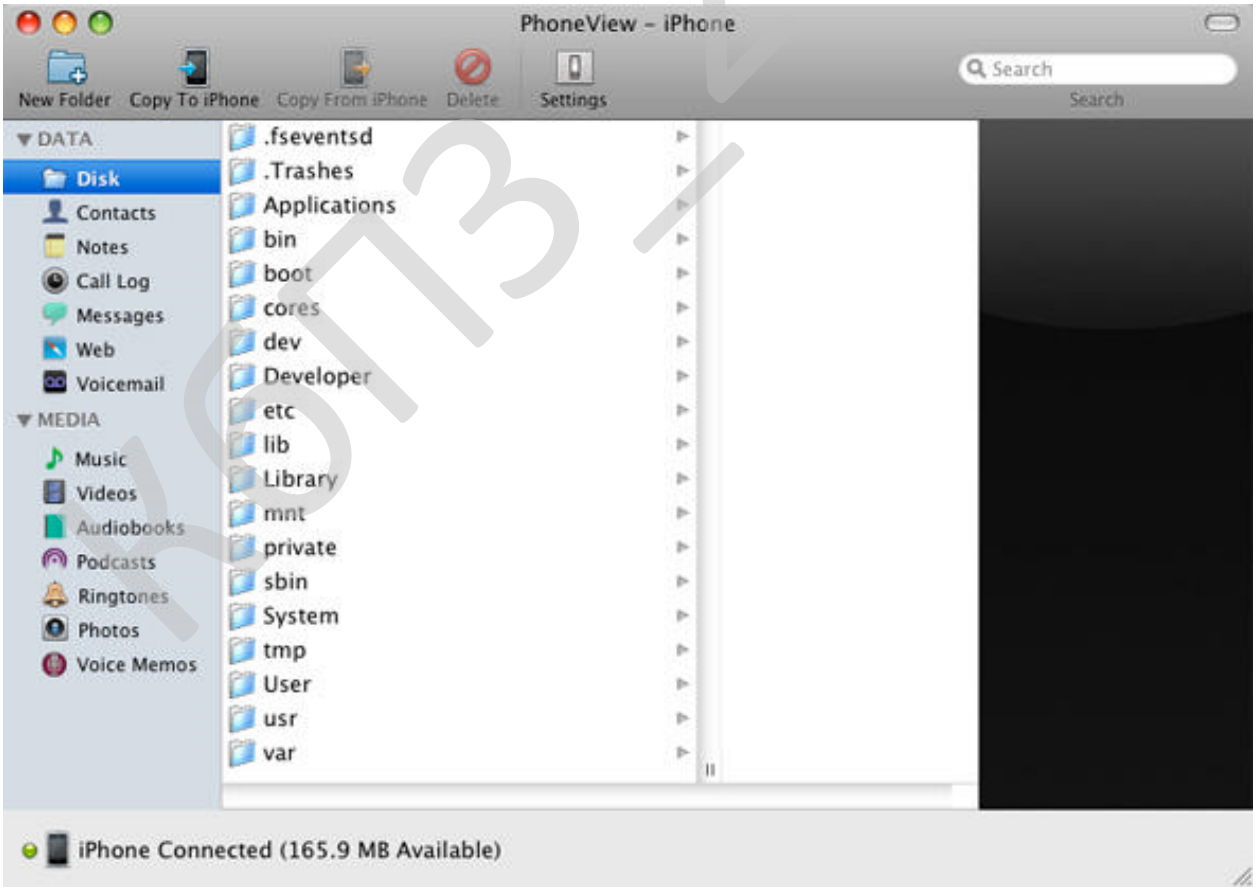

Рисунок 2.4 – Інтерфейс користувача PhoneView

На мій погляд, PhoneView – кращий файловий менеджер…і не тільки. Чому «не тільки»? Тому що додаток PhoneView уміє, крім роботи з файлами, переглядати й редагувати журнал дзвінків і SMS повідомлень, контакти адресної книги, замітки, копіювати фотографії, подкасти, пісні, відео та інші медіафайли.

У принципі, за допомогою PhoneView можна перетворити свій iPhone або iPod Touch у флешку. Але обов'язковою умовою є наявність операційної системи Mac OS X, версії PhoneView для Windows немає.

Як я вже писав, PhoneView уміє працювати й з вашими контактами.

Цікаво, що за допомогою PhoneView можна одержати доступ до вашої історії й закладок Safari на iPhone.

Легкий доступ до музики, фільмам, книгам, подкастам і іншому медіаконтенту iPhone. Подвійний клік по пісні почнеться її відтворення. Тут же можна одним кличем відправити все в iTunes.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програмне забезпечення написане із застосуванням мови Objective-C. Objective-C, відомий також як Objective C, Obj або Obj-C – компилюєма об’єктно-орієнтована мова програмування корпорації Apple, побудована на основі мови C й парадигм Smalltalk.

На відміну від C++, мова Objective-C повністю сполучимо із C (мова Objective-C є варіацією мови C) і код на C компілюється. Об'єктна модель побудована в стилі Smalltalk, тобто об'єктам посилають повідомлення.

Компілятор Objective-C входить в GCC і доступний на більшості основних платформ. Мова використовується в першу чергу для Mac OS X (Cocoa) і GNUstep – двох реалізацій об’єктно-орієнтованого інтерфейсу OpenStep.

Однією з відмітних рис Objective-C є його динамічність – цілий ряд рішень, звичайно прийнятих на етапі компіляції, тут відкладається безпосередньо до етапу виконання.

Ще однієї з особливостей мови є те, що він message-oriented у той час як C++ – function-oriented. Це значить, що в ньому виклики методу інтерпретуються не як виклик функції (хоча до цьому звичайно все зводиться), а саме як посилка повідомлення (з ім'ям і аргументами) об'єкту, подібно тому, як це відбувається в Smalltalk.

Такий підхід дає цілий ряд плюсів – так будь-якому об'єкту можна послати будь-яке повідомлення. Об'єкт може замість обробки повідомлення просто переслати його іншому об'єкту для обробки (так зване делегування), зокрема саме так можна легко реалізувати розподілені об'єкти (тобто об'єкти які знаходяться в різних адресних просторах і навіть на різних комп'ютерах).

Прив'язка повідомлення до відповідної функції відбувається безпосередньо на етапі виконання.

Мова Objective-C підтримує нормальну роботу з метаінформацією – так в об'єкта безпосередньо на етапі виконання можна запитати його клас, список методів (з типами переданих аргументів) і instance-змінних, перевірити, чи є клас нащадком заданого й чи підтримує він заданий протокол і т.п.

У мові є нормальна підтримка протоколів (тобто поняття інтерфейсу об'єкта й протоколу чітко розділені). Для об'єктів підтримується спадкування (не множинне), для протоколів підтримується множинне спадкування. Об'єкт може бути успадкований від іншого об'єкта й відразу декількох протоколів (хоча це скоріше не спадкування протоколу, а його підтримка).

На даний момент мова Objective-C підтримується компіляторами gcc і llvm (під керуванням Windows використовується в складі MinGW або cygwin).

Досить багато в мові перенесене на runtime-бібліотеку й сильно залежить від неї. Разом з компілятором gcc поставляється мінімальний варіант такої

бібліотеки. Також можна вільно скачати runtime-бібліотеку компанії Apple: Apple's Objective-C runtime.

Ці дві runtime-бібліотеки досить схожі (в основному відмінність полягає в іменах методів), далі приклади будуть орієнтуватися на runtime-бібліотеку від компанії Apple.

У мові Objective-C для позначення об'єктів використовується спеціальний тип id. Змінна типу id фактично є покажчиком на довільний об'єкт. Для позначення нульового покажчика на об'єкт використовується константа nil.

При цьому замість id можна використовувати й більше звичне позначення з явною вказівкою класу. Зокрема останнє дозволяє компіляторові здійснювати деяку перевірку підтримки повідомлення об'єктами – якщо компілятор з типу змінної не може зробити вивід про підтримку об'єктом даного повідомлення, то він видасть попередження.

Тим самим мова підтримує перевірку типів, але в нестрогій формі (тобто знайдені невідповідності вертаються як попередження, а не помилки).

На відміну від мови C++ посилка повідомлення nil є законною операцією, що завжди повертає нульове значення (nil).

Мова Objective-C дозволяє постачати мітками кожний аргумент, що помітно підвищує читаність коду й знижує ймовірність передачі неправильного параметра. Також підтримується можливість передачі довільної кількості аргументів у повідомленні.

Як і функції, повідомлення можуть повертати значення, при цьому на відміну від мови C, типом значення, що повертається за замовчуванням, є id.

Результат одного повідомлення можна відразу ж використовувати в іншому повідомленні:

Як уже говорилося, в Objective-C класи самі є об'єктами. Основним завданням таких об'єктів (називаних class objects) є створення екземплярів даного класу (це дуже схоже на паттерн Abstract Factory).

При цьому саме ім'я класу відіграє подвійну роль – з однієї сторони воно виступає як тип даних (тобто він може бути використаний для опису покажчиків на об'єкти даного класу). А з іншої сторони ім'я класу може виступати як об'єкт, якому посилає повідомлення (у повідомленнях ім'я класу може брати участь тільки як receiver).

У мові Objective-C немає убудованого типу для булевських величин, тому звичайно такий тип уводиться штучно. Далі для логічних величин буде використовуватися тип BOOL з можливими значеннями YES і NO (як це робиться в операційних системах NextStep, Mac OS X).

Як і в C++ опис класу і його реалізація розділені (звичайне опис міститься в заголовні файли з розширенням h, а реалізації – у файли з розширенням m).

У версії runtime від Apple всі класи мають загального предка – клас NSObject, що містить цілий ряд важливих методів.

Опис змінних нічим не відрізняється від опису змінних у структурах у мові C.

Описи ж методів помітно відрізняються від прийнятих у C++ і дуже сильно схожі на описи методів у мові Smalltalk.

Кожний опис починається зі знака плюс або мінус. Знак плюс позначає, що даний метод є методом класу (тобто його можна посилати тільки class object, а не екземплярам даного класу). Фактично методи класу є аналогами статичних методів у класах у мові C++.

Знак мінус служить для позначення методів об'єктів – екземплярів даного класу. Зверніть увагу, що в Objective-C всі методи є віртуальними, тобто можуть бути перевизначені.

Мова Objective-C дозволяє для аргументів методу задавати також один з наступних описувачей – oneway, in, out, inout, bycopy і byref. Дані описувачі служать для завдання напрямку передачі даних і способу передачі. Їхня наявність помітно спрощує реалізацію й роботу з розподіленими об'єктами.

У мові Objective-C можна по селекторі методу одержати адресу його функції, що реалізує (саме як функції мови C).

Мова Objective-C містить повноцінну підтримку протоколів (у C++ це абстрактний клас, що також іноді прийнятий називати інтерфейсом). Протокол являє собою просто список описів методів. Об'єкт реалізує протокол, якщо він містить реалізації всіх методів, описаних у протоколі.

Протоколи зручні тим, що дозволяють виділяти загальні риси в різнорідних об'єктів і передавати інформацію про об'єкти заздалегідь невідомих класів.

У самій мові Objective-C немає спеціальних команд для створення й знищення об'єктів (подібних new і delete). Це завдання лягає на runtime-бібліотеку й реалізуються за допомогою механізму посилки повідомлень.

Реально використовуваної й найбільше широко розповсюдженою схемою створення й знищення об'єктів в Objective-C є використовувана в операційних системах NextStep і Mac OS X.

Створення нового об'єкта розбивається на два кроки – виділення пам'яті й ініціалізація об'єкта. Перший крок реалізується методом класу alloc (реалізованому в класі NSObject), що виділяє необхідну кількість пам'яті (даний метод використовується для виділення пам'яті не тільки для об'єктів класу NSObject, але й будь-якого успадкованого від нього класу). При цьому в атрибут isa записується покажчик на class object відповідного класу.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи навігації та моніторингу файлової системи iOS.

В процесі розробки випускної кваліфікаційної роботи за другим

(магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Структура папок і файлів iPhone

Власники джейлбрейкнутих iPhone мають доступ до файлової системи свого апарата, що відкриває перед ними додаткові можливості – змінювати налаштування, недоступні офіційним користувачам, установлювати зламані програми й т.д. По суті, працювати із джейлбрейкнутим iPhone можна як із флешкою. Тільки от файлова система його не так проста. І часом досить складно знайти потрібний файл. У цьому розділі з усім цим розберемося.

Почнемо із установлених додатків. На прошиваннях 1.x.x з ними все було набагато простіше. Програми попадали в private/var/mobile/Applications і залишалося тільки знайти папку виду Proga.app. Але це було ще до відкриття App Store, коли програми завантажувалися з Installer. Нині цей пакетний менеджер втратив свою актуальність. Причому справа не тільки у відкритті онлайн-магазину від Apple. Для установки сторонніх додатків, що не потрапили в App Store, зараз набагато переважніша Cydia.

Втім, у цьому випадку це не так важливо. Програми із Cydia й Installer попадають в одну папку із передвстановленими додатками. Тільки тепер шлях до неї небагато інший – private/var/stash/Applications.Gg5Ly. При цьому набір букв і цифр, що випливає за словом Applications для кожного апарата різний.

Ще складніше із програмами, придбаними в App Store. Вони попадають у папку private/varmobile/Applications, але відкривши її, ви побачите купу додаткових папок з назвами виду: 2BF2F 8878-4846-B6E 7-019D0B5AE9B5. Шукати потрібну програму вам доведеться, клкаючи на них по черзі. У кожній з них є папка з ім'ям додатка, що ви шукайте:

– Стандартні додатки перебувають в /Applications.

– Додатки з App Store перебувають у папці /private/var/mobile/Applications.

– Фото з камери перебувають у папці:

/private/var/mobile/Media/DCIM/100APPLE.

– AddressBook, SMS і Mail розташовані в папці /private/var/mobile/Library/

– Теми розташовані в папці /Library/Themes.

Якщо ви не хакер, то залазити в неї вам, швидше за все, і не знадобиться, а от звернути увагу на сусідню папку Documents іноді може бути корисно. У ній зберігаються файли, які можуть бути цікаві. Скажемо, диктофони зберігають тут аудіо-записи, інтернет-пейджери – відправлені файли, а програми для роботи з документами – ті файли, які ви в них записали й т.д.

З відео, піснями й фотографіями справа простіша. Пісні зберігаються за адресою private/var/mobile/MediaiTunes_Control/Music. Правда, вам ще прийдеться їх пошукати по папках виду F00,F01,F02 і т.д. А шляхи для відео й фото наступні:

– Фотографії: private/var/mobile/Media/DCIM/100APPLE.

– Аудіозаписи диктофона: private/var/mobile/Media/Recordings.

– iPhoneVideoRecorder: private/var/mobile/Mediai/PhoneVideoRecorder.

– Cycorder: private/var/mobile/Media/Videos.

Якщо ви встановили нове прошивання, то буде набагато корисніше, якщо ви настроїте телефон як новий, а відновлювати дані будете не через резервне копіювання, а вручну, щоб не залишилися баги старого прошивання. Для цього вам потрібно знати, де зберігається ваша особиста інформація:

– Контакти: private/var/mobile/Library/AddressBook (у папці два файли, потрібно зберегти обидва).

– СМС: private/var/mobile/Library/SMS.

– Пошта: private/var/mobile/Library/Mail.

– Замітки: private/var/mobile/Library/Notes.

– Календар: private /var/mobile/Library/Calendar.

– Safari (закладки й історія): private /var/mobile/Library/Safari.

– Історія дзвінків: private /var/mobile/Library/CallHistory.

Якщо ви хочете встановити на свій iPhone оффлайн карту вашого міста, то файл із ім'ям MapTiles.sqlitedb потрібно залити в директорію private/var/mobile/Library/Caches/Maps/MapTiles (у залежності від версії прошивання), а файл для пошуку вулиць Bookmarks.plist у директорію private/var/mobile/Library/Maps. Не забудьте, що перед цим потрібно хоча б раз відкрити додаток "Карти" на вашім місті.

У висновку приведемо ще кілька шляхів, які можуть вам знадобиться:

– Рингтони – private/var/stash/Ringtones.tQjbOo.

– Шпалери – private/var/stash/Wallpaper.kPDEMH.

– Файли настроювань – private/var/mobile/Library/Preferences.

– Налаштування Wi-Fi:

private/var/preferences/SystemConfigurationcom.apple.wifi.plist,

– Завантажені файли – private/var/mobile/Library/Downloads,

private/var/mobile/Documents.

– Файли Кукі – private/var/root/Library/Cookies.

– MobileInstallation:

System/Library/PrivateFrameworks/MobileInstallation.framework

– Посилання "Додому" (збереження на робочий стіл) розміщаються в папці:

/private/var/mobile/Library/WebClips

Для джайлбрейкнутого телефону адреса папок буде починатися з /private/var/stash, тобто тут у джайбрейкнутому телефоні зберігаються стандартні додатки, шпалери й теми, рингтони, програми з Cydia.

3.2 Розробка структурної схеми

Структурна схема розробленої системи зображена на рисунку 3.1. На ній показано структуру розробленого програмного забезпечення системи навігації та моніторингу файлової системи iOS.

Структурно розроблена система навігації та моніторингу файлової системи iOS складається з наступних основних блоків:

– Операційна система IOS.

– Система навігації та моніторингу файлової системи iOS.

– Пам’ять.

– Комунікації.

– Вбудовані додатки.

Операційна система IOS складається чотирьох рівнів абстракції, розглянутих нижче, а також наступних структурних блоків:

– Jailbreak – API доступу до файлової системи.

– Трьохмірний графічний інтерфейс користувача Cover Flow.

– Інтерфейс користувача Cocoa Touch з мультисенсорним екраном.

– Технологія Multitouch.

Технологія iPhone представлена у вигляді шарів. Основний шар – це Core OS. На його вершині перебуває шар Core Services. На вершині шару Core Services перебуває шар Media. І на самій вершині перебуває шар Cocoa Touch. Взагалі можна ще більше спростити цю технологію. Можна розділити й об'єднати їх в 2 шари – це шар мови C і шар Cocoa мови Objective C.

Пам’ять буває наступних видів:

– RAM.

– Flash.

Комунікації використовуються наступні:

– Wi-Fi.

– Bluetooth.

– USB 3.0.

– GSM/EDGE.

– GPS, A-GPS.

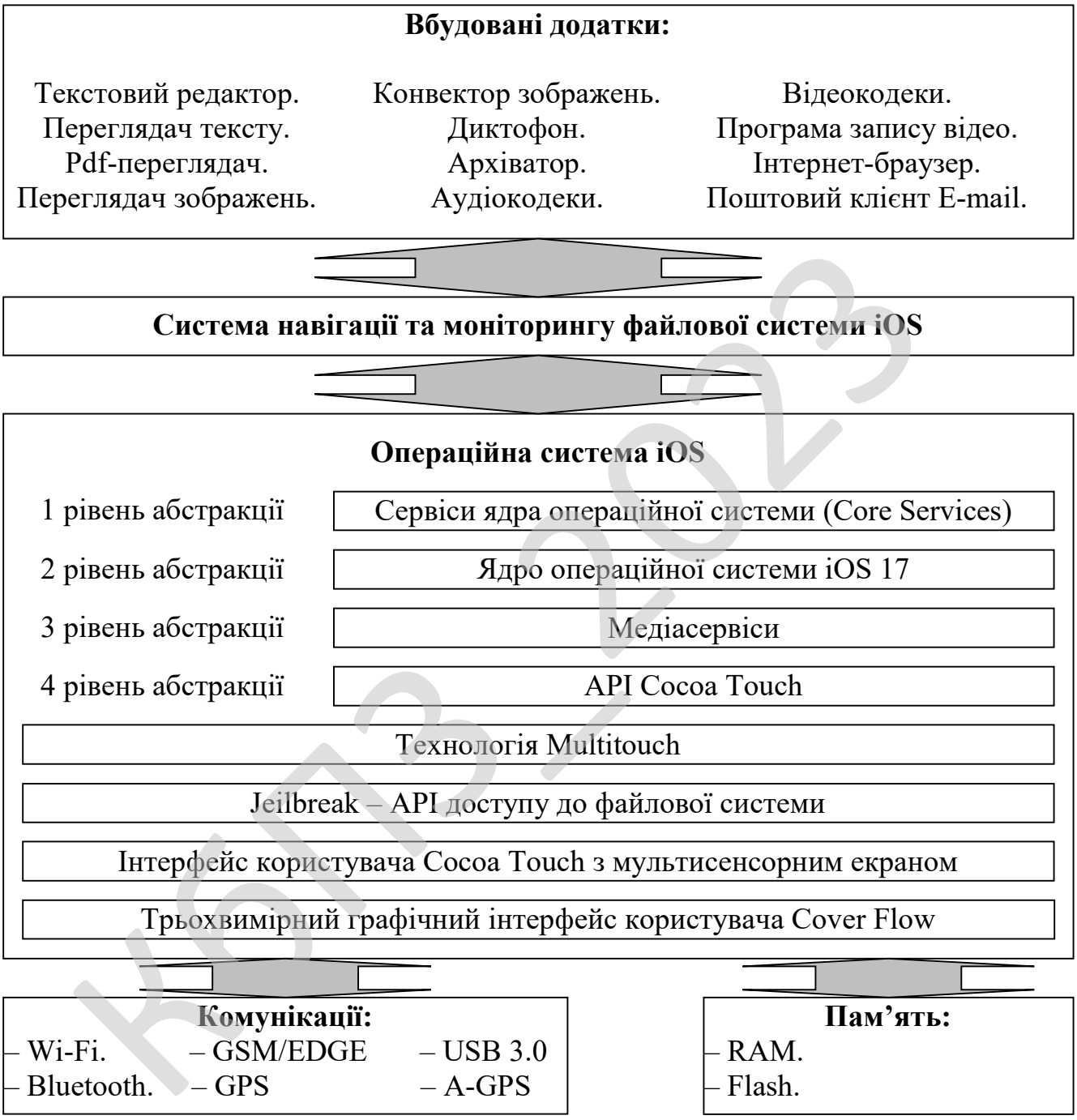

Рисунок 3.1 – Структурна схема системи

Джейлбрейк (злом) – це процес використання вразливостей заблокованого пристрою для встановлення програмного забезпечення, відмінного від наданого

виробником цього пристрою. Джейлбрейк дозволяє власнику пристрою отримати повний доступ до операційної системи та всіх функцій. Цей процес також називається зломом, оскільки він має на увазі "звільнення користувачів з в'язниці обмежень" пристрою.

Термін «джейлбрейк» найчастіше використовується щодо iPhone, оскільки iPhone вважається «заблокованим» мобільним пристроєм з наявних на ринку. У ранніх версіях iPhone не було магазину програм, і інтерфейс iOS вважався більш обмеженим для користувачів, ніж зараз. У США перша версія iPhone була доступна тільки в мережі оператора AT&T, і користувачі, які хотіли перейти до інших операторів зв'язку, не могли цього зробити, не зламавши iPhone.

Підхід Apple до програмного забезпечення характеризується як закрита екосистема телефону, тоді як Android доступна налаштування безлічі опцій. Основна мотивація багатьох хакерів – зробити iOS більш схожим на Android. Джейлбрейк був і залишається способом встановлення додатків, не схвалених Apple, а також способом налаштування інтерфейсу.

З моменту появи термін «джейлбрейк» також використовувався для визначення адаптації коду на інших пристроях, від телефонів до ігрових консолей. Іноді він використовується для позначення встановлення спеціального програмного забезпечення на мобільні пристрої або зняття обмежень на управління цифровими правами (DRM) для перегляду фільмів. Однак зазвичай термін "джейлбрейк" відноситься до продуктів Apple. Крім iPhone, він також застосовується до iPad та iPod Touch.

Термін "джейлбрейк" іноді використовується як синонім термінів "злом" (щодо програмного забезпечення) та "рутинг" (щодо телефонів). Рутінг можна описати як "джейлбрейк для Android", оскільки він спрямований на обхід засобів захисту, встановлених виробниками, для встановлення альтернативних мобільних операційних систем. Також часто зламують мережеві медіаплеєри Amazon Fire Stick та Roku для запуску мультимедійного програмного забезпечення замість вбудованих додатків та комутатори Nintendo для запуску емульованих ігор.

При зломі основні функції пристрою не змінюються. Зі зламаного iPhone або iPad, як і раніше, можна купувати і завантажувати програми з App Store. Однак для завантаження програм, відхилених Apple, та для використання додаткових функцій, отриманих внаслідок злому, використовуються незалежні магазини програм. Найпопулярнішим із них є Cydia – магазин для зламаних iOS-пристроїв, який зазвичай встановлюється у процесі злому.

Код злому зазвичай надається безкоштовно на форумах і сайтах спільнот зломщиків, які просувають необмежену користування пристроями. Більшість кодів злому супроводжується докладними інструкціями та інструментами застосування, але корисно мати деякі технічні знання.

Щодо джейлбрейка іноді використовуються терміни «прив'язаний» та «неприв'язаний».

– **Прив'язаний джейлбрейк** вимагає, щоб iOS-пристрій було підключено до комп'ютера під час увімкнення. Якщо не підключений до комп'ютера iPhone завантажуватиметься за допомогою спеціальної програми, він не перейде в стан злому.

– Для **відв'язаних джейлбрейків** комп'ютер не потрібен. Все необхідне переходу в стан злому міститься на iPhone. Перезавантаження без підключення до комп'ютера не впливає на джейлбрейк.

Зараз, коли програми для iPhone мають більший доступ до операційної системи, джейлбрейк став менш популярним. Оскільки Apple публічно не схвалює джейлбрейк, було впроваджено апаратні та програмні виправлення вразливостей, що використовуються під час злому. В результаті багато версій iOS не вдасться зламати швидко чи легко.

Чи легальний джейлбрейк?

Строго кажучи, джейлбрейк не є незаконним, але закони у всьому світі різняться, змінюються і часто не є однозначними, коли йдеться про джейлбрейк. Джейлбрейк або рутинг телефону є законним, якщо робиться для встановлення легально придбаних програм. Однак якщо його зроблено для встановлення

незаконно придбаних додатків, то й сам процес стає незаконним.

У США джейлбрейк підпадає під дію Закону про захист авторських прав у цифрову епоху, який торкається питань авторського права на цифрові матеріали. Розділ 1201 закону забороняє обхід цифрового блокування доступу до матеріалів, захищених авторським правом, у тому числі до програмного забезпечення. Конгрес переглядає закон кожні кілька років та поступово розширює список винятків. У 2010 році дозволили зламувати телефони, у 2015 році – смарт-годинник та планшети. З того часу до списку винятків було додано більше пристроїв; список продовжує розширюватися у міру перегляду.

Закони можуть відрізнятися у різних країнах світу. У багатьох країнах Джейлбрейк ніколи не обговорювався в суді, тому точна правова позиція залишається неясною.

Хоча Apple не підтримує джейлбрейк, компанія зазвичай не загрожує застосуванням юридичних заходів до зломщиків. Apple навіть відомий своєю подякою співтовариствам, які займаються джейлбрейком, за виявлення слабких місць у системі безпеки.

Незалежно від закону, при зламі телефону гарантія анулюється, тому якщо при джейлбрейку щось піде не так, розраховувати буде нема на що. Джейлбрейк також робить пристрій схильним до цілого ряду проблем безпеки, описаних нижче.

Чи безпечний джейлбрейк?

Джейлбрейк телефону легальний, але не завжди безпечний. В результаті джейлбрейку телефону у кіберзлочинців з'являються можливості його злому.

При джейлбрейку телефону відмова від системи безпеки Apple. Програми, завантажені зі сторонніх джерел, не перевіряються в Apple App Store і тому становлять загрозу безпеці. Після джейлбрейку телефон перестане отримувати оновлення iOS, включаючи оновлення безпеки Apple, що робить його більш вразливими для загроз безпеки.

Apple вважає джейлбрейк iOS порушенням умов використання та

інформує клієнтів, що це наражає телефон на наступні ризики:

– Уразливості у системі безпеки.

– Нестабільна робота.

– Можливі збої та зависання.

– Зменшення терміну служби батареї.

Тому Apple застерігає від джейлбрейків iPhone або інших пристроїв iOS. iPhone рідко заражається вірусами, але якщо це відбувається, причиною часто є злом телефону. Якщо щось трапиться з телефоном, його доведеться робити самостійно, оскільки джейлбрейк телефону анулює гарантію.

Слід враховувати, кому належить пристрій та яка інформація зберігається на телефоні. Наприклад, чи належить телефон роботодавцю? Чи синхронізується на телефоні робоча пошта? Будь-яка шкідлива діяльність ризикує не лише ваші дані, а й дані організації. Оскільки безпека зламаного телефону знижена, організація наражається на більший ризик кібератаки.

Організації, що надають співробітникам мобільні пристрої, зазвичай вживають заходів безпеки, що не дозволяють користувачам наражати на ризик дані компанії. Це може бути блокування телефону, яке дозволяє додавати або змінювати лише певні функції, підтримка пристроїв та програм в актуальному стані, встановлення агента мобільного пристрою, здатного виявити джейлбрейк.

Переваги джейлбрейку

Більше контролю за власним пристроєм

Apple прагне надати користувачам єдиний інтерфейс. Деякі користувачі вважають це обмеженням і хочуть персоналізувати свій телефон, додавши власні іконки, шпалери та меню. Джейлбрейк – це вирішення такого завдання. Після джейлбрейка ви, а не Apple або будь-хто, станете адміністратором свого пристрою з усіма відповідними правами. Наприклад, ви можете додавати іконки на домашній екран iPhone або встановлювати власні заставки. Джейлбрейк також розширює доступ до файлової системи та навіть розблокує можливість підключення інших пристроїв, що дозволяє підключити iPad до комп'ютера та

забезпечує більший контроль.

Встановлення та використання неавторизованих додатків

Apple забороняє завантаження різних програм у свій магазин App Store з міркувань безпеки. Джейлбрейк дозволить встановлювати програми, яких немає в App Store. Cydia – найпопулярніший магазин програм для зламаних телефонів, до якого можна додавати неавторизовані програми, такі як ігри та мережеві інструменти. Емулятори ретро-ігор також є гарним прикладом: Apple забороняє їх завантаження до свого магазину додатків (оскільки вони дозволяють грати в старі комп'ютерні ігри, не купуючи оригінальні копії). Однак вони знаходяться у вільному доступі до Cydia.

Видалення встановлених програм

iOS не дозволяє змінювати або видаляти встановлені за промовчанням програми, такі як Apple Watch, Погода, Ігри та інші. Ці програми займають місце в пам'яті, що незручно для людей, які не користуються ними. Джейлбрейк дозволяє видалити встановлені за промовчанням програми Apple і використовувати замість них сторонні програми. Це дозволить, наприклад, налаштувати голосовий помічник Сірі на використання Google Maps замість Apple Maps.

Доступ до додаткових функцій захисту від крадіжок

Деякі користувачі вважають, що джейлбрейк надасть їм доступ до покращених функцій захисту від крадіжки. Наприклад, у iPhone є функція «Знайти iPhone», але вона не працює, коли телефон перебуває в режимі польоту, вимкнено або не ловить мережу. Існують програми для зламаних пристроїв, які, як стверджують, працюють краще, ніж функція «Знайти iPhone», наприклад iCaughtU. Коли зловмисник вводить неправильний пароль, передня камера фотографує його та надсилає фото власнику пристрою електронною поштою.

Недоліки джейлбрейку

Припинення автоматичних оновлень

Більше не вдасться отримувати автоматичні оновлення безпосередньо від

Apple. Потрібно буде чекати, поки співтовариство зломщиків виконає джейлбрейк кожної версії iOS. Злом оновлень вимагає часу, крім того, доведеться зламувати кожну версію iOS, що випускається Apple. Це означає, що оновити зламаний телефон не вдасться доти, доки не буде зламано останнє оновлення, що може статися не відразу. Виконання джейлбрейку після великих оновлень може виявитися досить проблематичним. Чи вартує джейлбрейк цих труднощів?

Неможливість інсталювати деякі програмні оновлення

Внаслідок деяких несанкціонованих змін iPhone може назавжди втратити працездатність після встановлення оновлень iOS, що постачаються Apple.

Анулювання гарантії на телефон

Apple заявляє, що несанкціонована зміна iOS є порушенням ліцензійної угоди для iOS. Через це Apple може відмовити в обслуговуванні iPhone, iPad або iPod touch, на якому встановлено неавторизоване програмне забезпечення. Таким чином, якщо в результаті джейлбрейку пристрій виявиться пошкодженим або несправним, Apple може відмовити в будь-якому сервісному ремонті.

Зменшення терміну служби батареї

Зламування програмного забезпечення може призвести до прискореної розрядки батареї, що скорочує час роботи iPhone, iPad або iPod touch від однієї зарядки акумулятора.

Телефон може «перетворитися на цеглу»

Телефон перестане завантажуватися, реагувати на команди та виконувати дзвінки – «перетвориться на цеглу». Сам собою джейлбрейк не блокує телефон, але існує ризик повної відмови функціонування телефону.

Втрата доступу до контенту та сервісу

Часто причиною злому телефону є бажання отримати доступ до більшої кількості контенту, але іноді це може виявитися неефективним, оскільки можна втратити доступ до інших сервісів, таких як iCloud, iMessage, FaceTime, Apple Pay, Погода та Stocks. Сторонні програми, які використовують сервіс Apple Push Notification, мають проблеми при отриманні повідомлень або отримують

повідомлення, призначені для інших зламаних пристроїв. Інші сервіси, що використовують push-сповіщення, такі як iCloud та Exchange, стикаються з проблемами синхронізації даних з відповідними серверами. Існували повідомлення про те, що сторонні постачальники блокують зламані пристрої.

Збільшення ймовірності поломки телефону

Імовірність виходу з ладу зламаного iPhone або iPad може бути вищою. Програми, доступні для зламаних пристроїв, отримують доступ до функцій та API, недоступних для програм, схвалених Apple. Робота таких програм могла не тестуватися. Це може призвести до частих та несподіваних збоїв пристрою, збоїв та зависання вбудованих та сторонніх програм та втрати даних.

Ненадійна передача голосу та даних

Джейлбрейк може призвести до обриву дзвінків, повільних або ненадійних з'єднань для передачі даних, а також до затримок передачі даних про місцезнаходження або передачу неточних даних.

Витік даних

Сумно відомий інцидент зі зламаними пристроями стався, коли зловмисники отримали доступ до даних для входу в iCloud у 225 000 людей, які намагалися виконати джейлбрейк. Витоку сприяли вразливості системи безпеки, що утворилися в результаті джейлбрейку, що допомогло зловмисникам отримати доступ до пристроїв користувачів.

Проблеми з безпекою

Закритий характер iOS робить її однією з найбезпечніших мобільних операційних систем, оскільки забезпечується захист як особистої інформації, так і самої системи. Зламування телефону збільшує можливості зловмисників вкрасти особисту інформацію, пошкодити пристрій, атакувати мережу або впровадити шкідливі програми, шпигунські програми або віруси.

Ризики безпеки при джейлбрейку

Джейлбрейк телефону є загрозою безпеці. Джейлбрейк надає вам більше можливостей контролю над пристроєм, але також він надає більше можливостей

контролю за всіма програмами, які працюють на пристрої. Найбільші загрози безпеці виникають через дозволи цим програмам запитувати root-доступ на пристрої. Якщо на пристрої встановлені шкідливі програми, вони можуть отримати доступ до root, тобто повний доступ до всіх даних на пристрої.

Під час злому порушується закрита екосистема пристрою, що забезпечується Apple для захисту користувачів від загроз безпеки. Зламані телефони набагато сприйнятливіші до вірусів і шкідливих програм, оскільки дозволяють не виконувати перевірку програм Apple, що забезпечує завантаження програм без вірусів. Джейлбрейк допускає встановлення піратських програм, що розповсюджуються безкоштовно додатків та ігор. Це означає, що довіреними стають розробники всіх додатків, що встановлюються, а не тільки розробники Apple.

Дані з банківських програм, збережені паролі та дані облікових записів соціальних мереж можуть опинитися під загрозою, якщо ця інформація стане доступною зі зламаного iPhone. Цей ризик розкрився, коли шкідлива програма для злому iOS, KeyRaider, вкрала 225 000 ідентифікаторів Apple ID та тисячі сертифікатів, закритих ключів та чеків про покупки. Внаслідок жертви повідомляли про незвичайну історію покупок додатків з боку вкрадених у них облікових записів. В інших випадках телефони жертв були заблоковані для отримання викупу.

Крім високого ризику зараження шкідливими програмами, зламані iPhone часто містять помилки, які можуть призводити до збоїв телефону та відключення важливих функцій. Зі зростанням використання смартфонів зростає і ризик мобільних злочинів. Тому важливо бути в курсі останніх загроз та шахрайських схем, а також встановлювати комплексний мобільний захист на пристрої.

Як полагодити зламаний телефон

Зламаний телефон можна відремонтувати, просто відновивши iPhone. Не потрібно вручну видаляти встановлені програми злому, оскільки в результаті з iPhone буде видалено все, і пристрій буде скинуто до заводських налаштувань

Apple.

Перед початком переконайтеся, що ви здійснили повне резервне копіювання даних з iPhone або iPad. Це пов'язано з тим, що в процесі видалення джейлбрейка буде виконано повне очищення пристрою та відновлення до стандартної конфігурації. Тому необхідно заздалегідь створити резервну копію всіх файлів, які ви хочете зберегти. Найкраще зберегти файл резервної копії у двох місцях (локально та у хмарі).

Крок 1. Резервна копія в iCloud

– Підключіть iPhone, iPad або iPod touch до Wi-Fi.

– Перейдіть до меню Налаштування, виберіть [Ваше ім'я] та iCloud.

– Переконайтеся, що увімкнуто перемикач Резервна копія в iCloud.

– Натисніть кнопку Створити резервну копію та не відключайтеся від мережі Wi-Fi до завершення процесу.

Щоб перевірити хід виконання та підтвердити завершення резервного копіювання, перейдіть до меню **Налаштування**, виберіть **[Ваше ім'я]**, потім **iCloud** та **Резервна копія в iCloud.** Під кнопкою **Створити резервну копію** відображається **дата** та **час** створення останньої резервної копії.

Крок 2. Скасування джейлбрейку

1. Підключіть iPhone або iPad до комп'ютера або пристрою Mac за допомогою оригінального кабелю USB.

2. Запустіть iTunes на комп'ютері.

3. Розблокуйте пристрій та вимкніть функцію Знайти iPhone.

4. Перейдіть до меню Налаштування, виберіть [Ваше ім'я] та iCloud.

5. Переконайтеся, що перемикач Пошук iPhone вимкнено. Щоб вимкнути цю функцію, необхідно ввести Apple ID та пароль.

6. У iTunes на комп'ютері виберіть пристрій, коли він відобразиться.

7. На панелі Огляд натисніть кнопку Відновити. Запуститься процес видалення джейлбрейку.

8. Під час цієї процедури пристрій перезавантажиться. Пристрій запитає,

чи потрібно відновити дані з резервної копії. Можна вибрати опцію iCloud, якщо потрібно відновити дані з файлу, створеного раніше.

9. Після завершення процесу iOS-пристрій повернеться до заводських налаштувань. З'являться стандартні кроки налаштування, які ви виконували під час першого увімкнення пристрою.

Якщо з будь-яких причин не вдається відновити зламаний iPhone, використовуйте режим відновлення, щоб видалити дані з пристрою.

На закінчення: уразливості програм на зламаних пристроях дозволяють зловмисникам легко викрадати конфіденційні дані, такі як платіжна інформація. Увага до небезпек допомагає захиститися під час роботи в інтернеті.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.2. З рисунку видно, що розроблена система складається з наступних частин:

– Параметри системи управління файлами на iPhone.

– Система навігації та моніторингу файлової системи iOS.

– Відкриття файлів.

– Блок завантаження/вивантаження файлів.

Розглянемо більш детально кожний з блоків розробленої програми.

Параметри файлового менеджера складаються з наступних функціональних блоків:

– Конвертор зображень.

– Якість відео.

– Налаштування передачі файлів.

– Налаштування редактору текстових документів.

– Пароль.

– Вид.

– Налаштування інтернет-браузера.

– Аудіо налаштування.

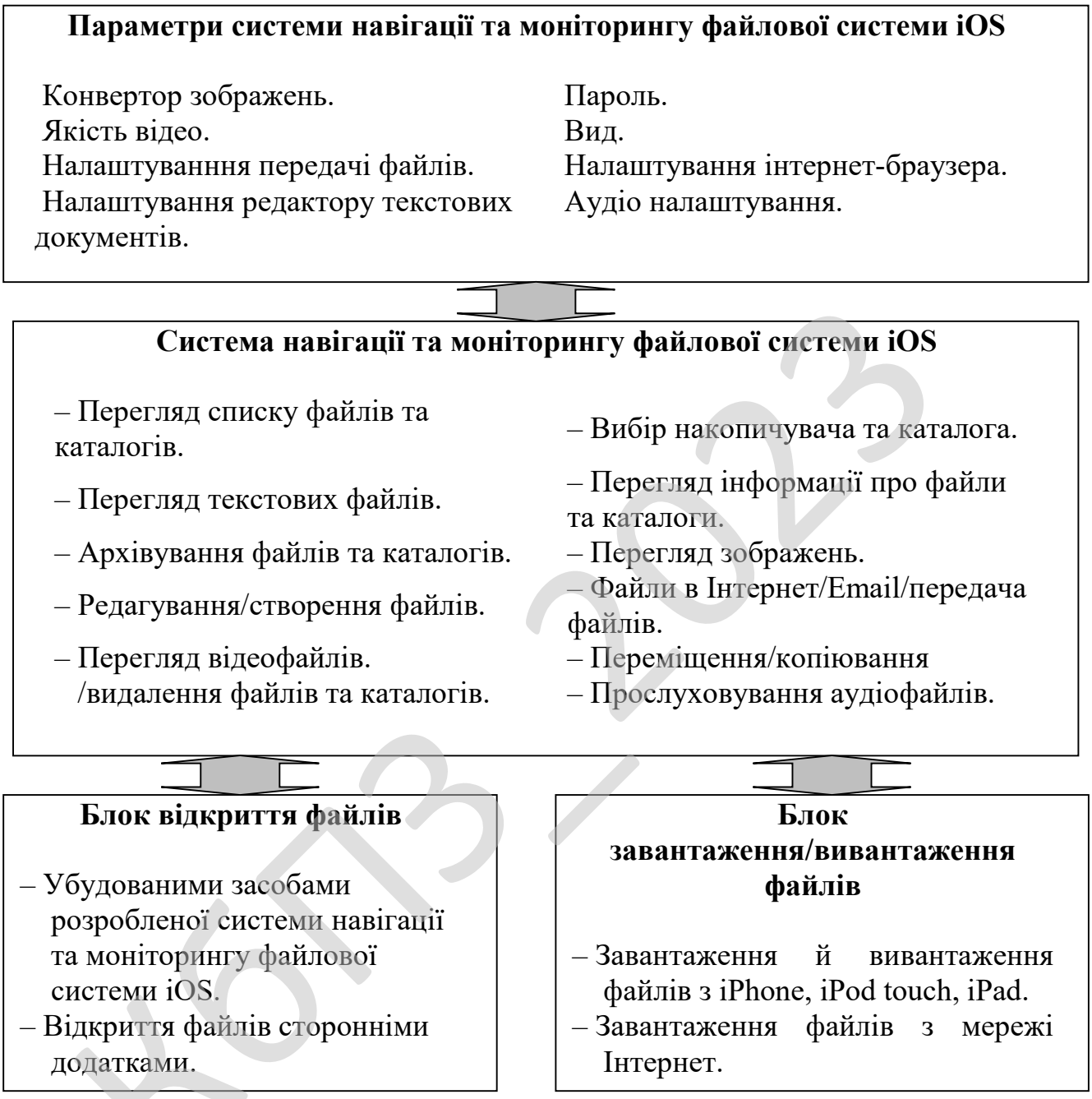

Рисунок 3.2 – Функціональна схема системи

Система навігації та моніторингу файлової системи iOS складається з наступних функціональних блоків:

– Перегляд списку файлів та каталогів.

– Вибір накопичувача та каталога.

– Перегляд текстових файлів.

– Перегляд інформації про файли та каталоги.

– Архівування файлів та каталогів.

– Перегляд зображень.

– Редагування/створення файлів.

– Файли в Інтернет/Email/передача файлів.

– Перегляд відеофайлів.

– Переміщення/копіювання/видалення файлів та каталогів.

– Прослуховування аудіофайлів.

Блок відкриття файлів:

– Убудованими засобами розробленої системи управління файлами на iPhone.

– Відкриття файлів сторонніми додатками.

Блок завантаження/вивантаження файлів складається з наступних функціональних блоків:

– Завантаження й вивантаження файлів з iPhone, iPod touch, iPad.

– Завантаження файлів з мережі Інтернет.

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

3.4 Розробка діаграми процесів

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

Після початку роботи розробленого ПЗ ми потрапляємо до інтерфейсу ПЗ звідки проходе взаємодія з бібліотекою роботи з файловою системою: Функції підключення до зовнішнього пристрою; Функції завантаження файлів; Функції обробки файлів та каталогів. Далі через обробник помилок ПЗ проходе виведення списку дисків та проводиться моніторинг файлової підсистеми. Через виведення

списку файлів та каталогів та вибору файлів/каталогів проходе: Видалення файлу; Перегляд файлу; Створення файлу; Передача файлу на зовнішній пристрій; Копіювання файлу; Перейменування файлу.

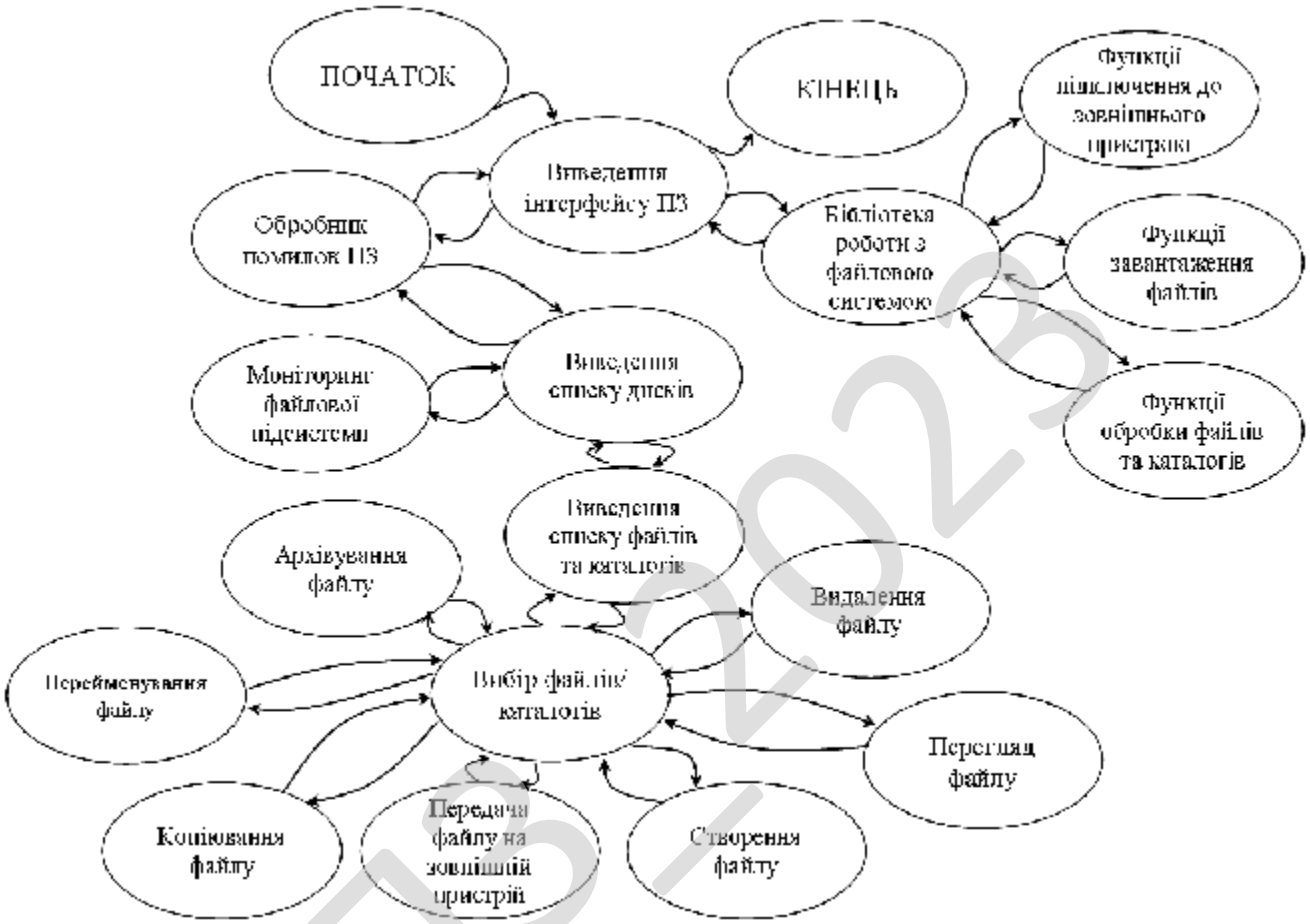

Рисунок 3.3 – Діаграма взаємодії процесів

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

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

– Виведення головного вікна ПЗ.

– Сканування та виведення списку дисків.

– Обрання диску.

– Виведення стану каталогів та властивостей диску.

– Запит змінити диску?

– Зміна накопичувача.

– Запит перегляду каталогу?

– Перехід у вказаний каталог.

– Виведення вмісту каталогу.

– Запит обрання файлу?

– Запит перегляду файлу?

– Відкриття файлу ПЗ вказаним ОС як основний обробник.

– Виведення вмісту обраного файлу на екран.

– Запит редагувати файл?

– Підпрограма редагування файлів.

– Запит копіювати файли?

– Копіювання вказаних файлів.

– Запит створення резервної копії?

– Створення резервної копії групи вказаних файлів.

– Запит видалити файл?

– Видалення обраних файлів.

– Збереження змін.

– Запит Exit?

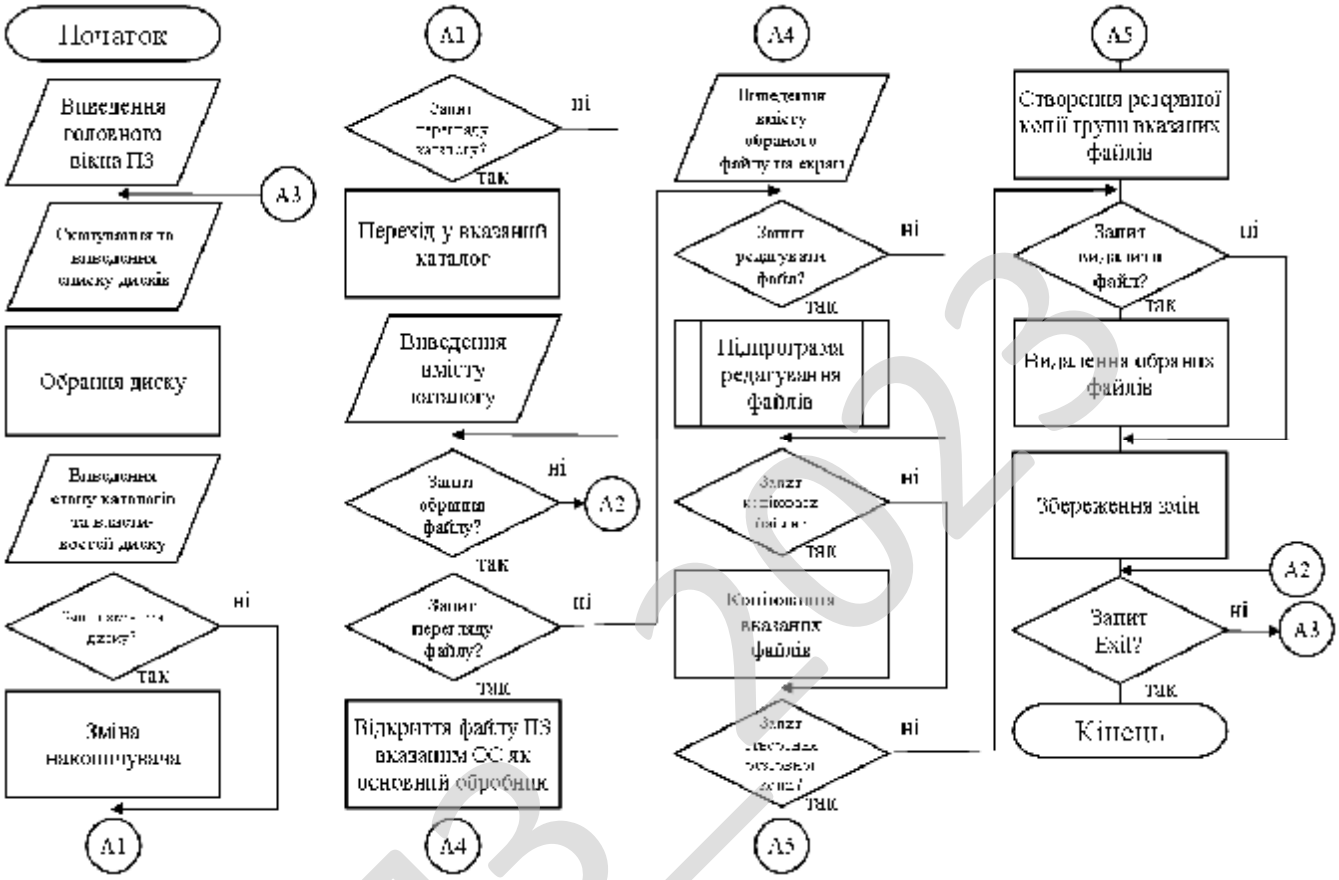

Рисунок 4.1 – Блок-схема основної програми

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

– Запит редагування текстового файлу?

– Завантаження текстового редактора з параметрами файлу.

– Запит створення аудіофайлу?

– Завантаження диктофону з параметром запису.

– Запит створення відеофайлу?

– Завантаження камери у режимі запису відео.

– Запит створення зображення?

– Завантаження камери у режимі фото.

– Запит редагування файлу зображення?

– Завантаження графічного редактору.

– Запит створення E-mail скрині?

– Завантаження поштового клієнта.

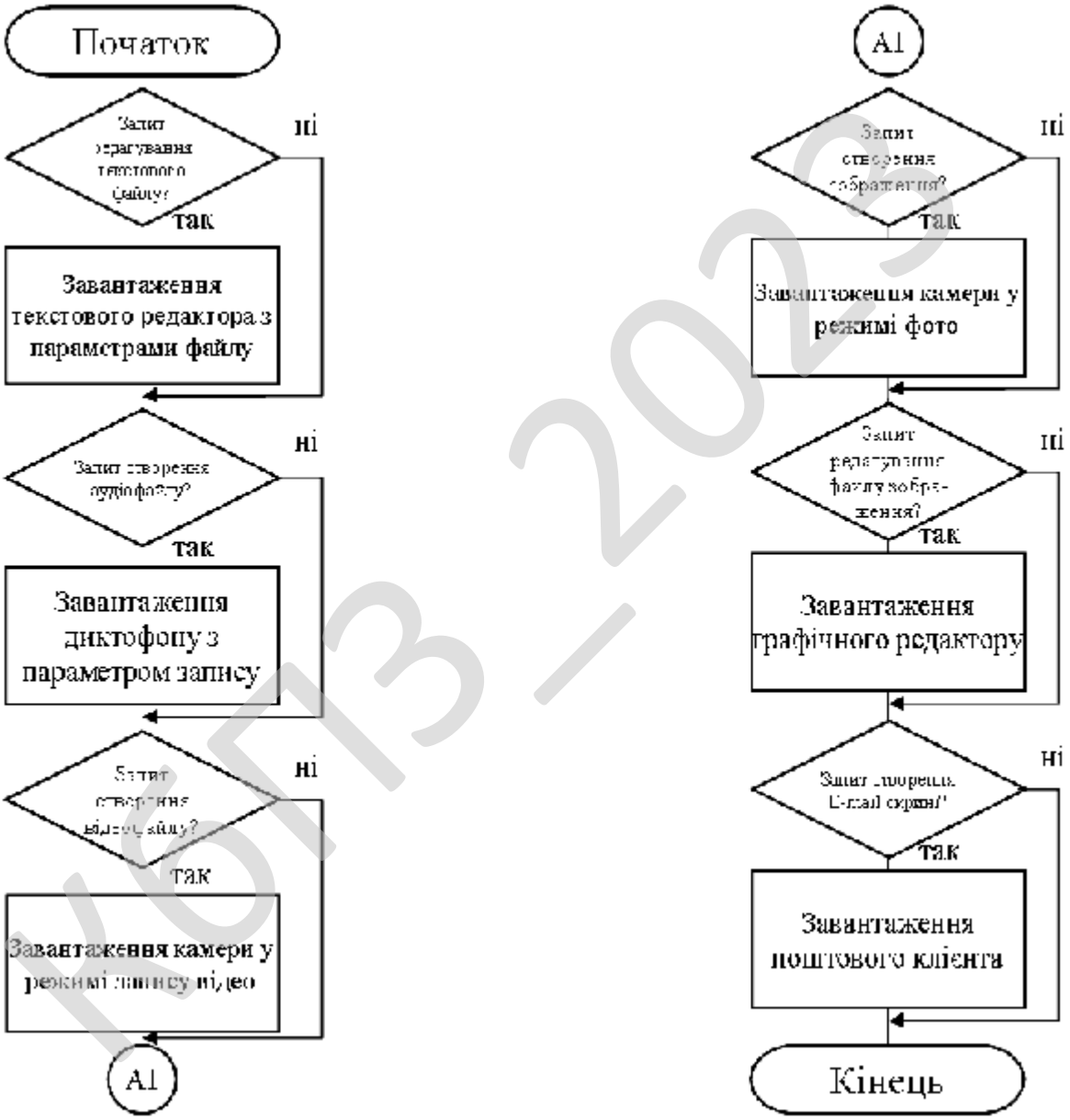

Рисунок 4.2 – Блок-схема підпрограми редагування файлів

Розглянемо, як реалізуються елементи розробленого програмного забезпечення управління файлами на iPhone, за допомогою мови Ojective-C.

Розширені класи

Однієї з речей, прямих аналогів якої немає в C++, є ідея категорії – колекції методів, які додаються до існуючого класу. Будучи завантаженої, категорія не має відмінностей від інших методів. Категорії, як і класи, оголошуються з інтерфейсом і реалізацією.

Інтерфейси категорії можуть бути використані без відповідної реалізації для розкриття інших методів, надаючи щось схоже на дружні функції в C++. Якщо ви реалізуєте методи в класі, але не повідомляєте їх в інтерфейсі, ви одержите попередження при компіляції в тих місцях, де ви використовуєте їх. Ви можете зробити такий інтерфейс категорії:

```
@interface AnObject (Private)
- (void) privateMethod;
@end
```

Якщо ви помістите це в початок файлу реалізації, ви не одержите попередження під час компіляції, коли відправите повідомлення -privateMethod екземплярам AnObject. Ім'я в дужках (Private) – просте ім'я категорії. Воно може бути кожним. Зверніть увагу на те, що без розділу @inmplementation, це просто стане раннім оголошенням С функції. Якщо немає відповідної реалізації в С, ви одержите помилку лінкувальника. Якщо немає відповідної реалізації методу в Objective-C, ви одержите виключення в рантаймі.

Ви можете надати додаткову реалізацію методу, використовуючи категорію точно таким же шляхом, як і коли ви надавали «нормальні» методи:

```
@implementation AnObject (NewMethods)
- (void) newMethod
{
  ...
}
@end
```

Якщо ви відправите повідомлення -newMethod будь-якому екземпляру AnObject, цей метод буде викликаний. Ви також можете використовувати це для

заміни існуючих методів в об'єкті на свою власну версію, тому вам не потрібний доступ до об'єкта. Цей підхід часто використовується для доповнення різних методів бібліотечних класів додатковим функціоналом, а також може бути використаний для виправлення багів у сторонніх бібліотеках, вихідних кодів яких у вас немає.

Менш документованою частиною категорій є те, що категорії дозволяють вам додавати відповідності протоколів до існуючих об'єктів. Якщо ваша категорія переймає протокол, ви одержите попередження під час компіляції, якщо ви не надаєте реалізацій кожного методу, і стане можливим провести тести під час виконання програми для перевірки відповідності. Ми використовували цю можливість в ?toil? для додавання відповідності колекції протоколів до всієї колекції класів в Foundation, давши їм сумісний інтерфейс.

Об'єктна модель

В C++ об'єкти – це структури з набором зв'язаних функцій, які є або статичними, або віртуальними. Статична функція – семантичний еквівалент функції в C зі схованим першим аргументом, що містить об'єкт. Це вкрай неефективне розширення над C, так як наступні речі еквівалентні:

```
// C++
Object->function();
// C
function(Object);
```

Версія на C++ довшае, але не дає яких-небудь семантичних відмінностей.

C++ також підтримує дещо більше розумне у формі віртуальних функцій, що є членами класу. Якщо функція в попередньому прикладі є віртуальної, яка реальна функція буде викликана, залежить від класу Object. Якщо вона статична, то це буде залежати від того, об'єктом якого класу зухвалий уважає Object.

В Objective-C не було додано еквівалента статичної функції, що є членом класу. Якщо ви хочете таку функціональність, ви просто використовуєте функції на C. Методи в Objective-C схожі з віртуальними методами в C++, але з декількома важливими відмінностями. Перше – інший синтаксис:

```
[object message];
```

Цей код відправляє повідомлення з ім'ям message об'єкту. Який код буде викликаний як результат цієї дії, повністю залежить від класу об'єкта. От одна дуже важлива відмінність між цим і аналогом в C++: в C++ цей механізм дотепер небагато залежить від того, об'єктом якого класу ви вважаєте даний об'єкт.

Припустимо, що у вас є ієрархія із чотирьох класів. A – базовий клас, B і C – дочірні класи A, а D – дочірній клас B. Якщо кожний з них, крім класу A, реалізує віртуальну функцію doSomething() в C++, ви можете викликати її, тільки використовуючи шаблон. Розглянемо наступний рядок:

```
Object.doSomething();
```

Якщо ви припускаєте, що Object – це об'єкт класу B або D, і він дійсно є об'єктом D, буде викликана реалізація, певна в D. Якщо ви припускаєте, що це об'єкт класу C, він викликає реалізацію з C. Якщо ви думаєте, що це об'єкт класу A, вам необхідно буде спробувати провести два явних динамічних приведення й подивитися, який спрацює, або використовувати шаблон.

Якщо у вас є такий же порядок класів в Objective-C, із класами B, C і D, що реалізують метод doSomething, ви можете спробувати наступне:

```
[object doSomething];
```

Якщо ви припускаєте, що даний об'єкт є типом B, а насправді він типу C, метод doSomething однаково буде викликаний. Якщо ви думаєте, що це екземпляр класу A, при компіляції ви одержите попередження про те, що екземпляри класу A не відповідають на повідомлення doSomething. Якщо це дійсно екземпляр класів B, C або D, то це спрацює в рантаймі, а якщо ж він дійсно екземпляр класу A, ви одержите виключення в рантаймі.

Множинне спадкування недоступно в Objective-C, але тут воно набагато менш затребувано, чим в C++. В C++ метод пошуку заснований на ланцюжку спадкування, так що ви можете використовувати два класи ієрархічно, поки в них немає загального суперкласу. В Objective-C ви можете використовувати будь-які два класи, як повністю взаємозамінні, якщо вони відповідають на ті ж повідомлення.

Класи не є особливими. В Smalltalk класи – це просто об'єкти з деякими спеціальними можливостями. Те ж саме справедливо й в Objective-C. Клас – це об'єкт. Він відповідає на повідомлення так само, як і об'єкт. І Objective-C, і C++ розділяють виділення пам'яті для об'єкта і його ініціалізацію:

– В C++ виділення пам'яті для об'єкта робиться за допомогою оператора new. В Objective-C це робиться відправленням класу повідомлення alloc, що, у свою чергу, викликає malloc() або аналог.

– Ініціалізація в C++ відбувається за допомогою виклику функції з ім'ям, аналогічним ім'я класу. Objective-C не проводить розходжень між методами ініціалізації й інших методів, але за згодою метод ініціалізації за замовчуванням називається init.

Коли ви повідомляєте метод, на який відповідає об'єкт, оголошення починається зі знака «-», а «+» використовується для методів класу. Звичайно ці префікси використовуються для повідомлень у документації, так що ви повинні писати +alloc і -init, щоб указати, що alloc посилає повідомлення класу, а init – екземпляру.

Класи в Objective-C, як і в інших об’єктно-орієнтованих мовах, це «фабрики» об'єктів. Більшість класів самі не реалізують метод +alloc; замість цього вони успадковують його від супер-класса. В NSObject, базовому класі в більшості програм на Objective-C, метод +alloc викликає +allocWithZone. Він приймає NSZone як аргумент, – C-Структуру, що містить деяку лінію поводження для виділення пам'яті під об'єкти. Однієї з гарних можливостей, що з'явилися наслідком того, що семантика створення об'єкта визначена бібліотекою, а не мовою, є ідея кластера класів. Коли ви відправляєте об'єкту повідомлення -init, він повертає ініціалізований об'єкт. Це може бути об'єкт, якому ви відправили повідомлення (і звичайно так і є), але не обов'язково повинне бути саме так. Це ж вірно й для інших ініціалізаторів. Дозволяється мати спеціалізовані підкласи публічного класу, які більше ефективні для різних даних.

Розповсюджений трюк реалізації цієї можливості називається «isa-swizzling». Як я вже сказав, об'єкти в Objective-C є структурами C, де перший елемент – покажчик на клас. До цього елемента можна одержати доступ точно так само, як і до іншим змінним екземпляра; ви можете змінити клас об'єкта в рантаймі, просто привласнивши нове значення. Проте, у вас може бути суперклас, які визначає інтерфейс, а потім набір підкласів, які визначають поводження. Наприклад, ця техніка використовується в стандартному строковому класі (NSString), у якому є різні екземпляри для різних кодувань, для статичних рядків і т.д.

Так як класи є об'єктами, ви можете робити з ними майже все, що й з об'єктами. Наприклад, ви можете збирати їх у колекції. Я використовую цей формат досить часто, коли в мене є набір вхідних подій, які потрібно обробити екземплярами різних класів. Ви можете створити словник відповідностей іменам подій, а потім створювати новий об'єкт для кожного вхідної події. Якщо ви зробите це як бібліотека, це дозволить користувачам коду просто реєструвати власні оброблювачі.

Керування пам'яттю. Традиційно Objective-C не надає яких-небудь можливостей по керуванню пам'яттю. У ранніх версіях кореневий клас Object реалізовував метод +new, що викликав malloc() для створення нового об'єкта. Коли ви закінчували роботу з об'єктом, ви посилали повідомлення -free. OpenStep додав підрахунок посилань. Кожний об'єкт, наслідуваний від NSObject, відповідає на повідомлення -retain і -release. Коли ви хочете зберігати покажчик на об'єкт, ви відправляєте повідомлення -retain. Коли ви закінчуєте роботу, відправляєте повідомлення -release. Ця модель має одну невелику проблему. Часто ви не хочете зберігати покажчик на об'єкт, але й не хочете поки що його звільняти. Типовий приклад – повернення до об'єкта. Об'єкту, який викликає, може знадобитися зберігати покажчик на об'єкт, а вам немає. Рішенням цієї проблеми став клас NSAutoreleasePool. На додаток до -retain і -release, NSObject також відповідає на повідомлення -autorelease.

Коли ви відправляєте яке-небудь із них, воно реєструє себе з поточним пулом автоматичного вивільнення. Коли об'єкт пула знищений, відсилається повідомлення -release до кожного об'єкта, що раніше одержав повідомлення -autorelease. У додатках OpenStep екземпляр NSAutoreleasePool створюється на початку кожного циклу й знищується наприкінці. Також ви можете створювати власні екземпляри, щоб звільнити об'єкти, що вивільняються автоматично, швидше.

Цей механізм ліквідує багато копіювання, у якому перебуває C++. Також слід зазначити тут те, що в Objective-C мінливість – атрибут об'єкта, а не посилання. В C++ у вас константні покажчики й не константні покажчики. Вам не дозволяється застосовувати неконстантні методи до константного об'єкта. Це не гарантує те, що об'єкт не зміниться, – просто ви його не зміните.

В Objective-C загальний шаблон визначає незмінного класу, а потім змінюваного підкласу. NSString являє типовий приклад, – у нього є змінюваний підклас NSMutableString. Якщо ви одержуєте NSString і хочете зберегти його, ви можете послати повідомлення -retain і зберегти покажчик без операції копіювання. Як альтернатива ви можете послати NSString повідомлення +stringWithString:. Він перевірить, чи змінюваний аргумент і якщо так, поверне оригінальний покажчик.

Objective-C 2.0 як з рантайм Apple, так і GNU, підтримує не збирач, що переміщає, сміття, що рятує від необхідності користуватися повідомленнями -retain і -release. Це доповнення до мови це не завжди добре підтримується існуючими фреймворками й повинне використовуватися з обережністю.

Протоколи

В Objective-C протоколами є набори повідомлень, що реалізують клас. Ви можете вказати, що покажчик повинен указувати на клас, що реалізує даний інтерфейс:

```
id<AnInterface> object;
NSString<AnInterface> *string;
```

Перший приклад еквівалентний оголошенню змінної, як тип інтерфейсу в Java. В C++ найближчим еквівалентом є використання абстрактних класів замість інтерфейсів і вказівка абстрактного класу як тип.

Більше цікавий другий приклад. Строковій змінній дозволено бути будь-яким підкласом NSString, що реалізує AnInterface, що дозволяє вам обмежувати підмножину підкласів, що реалізує окремий інтерфейс. Загальний приклад:

```
NSObject<MyDelegateProtocol> *delegate;
```

Це дозволяє йому бути будь-яким підкласом NSObject (і тому методи, які ви очікуєте від будь-якого об'єкта, будуть працювати) і додатково потрібно реалізувати певний протокол. А от альтернативний варіант:

```
id <NSObject, MyDelegateProtocol> delegate;
```

Це працює для NSObject, так як NSObject є й класом, і протоколом, із класом, що переймає протокол. Це було зроблено так, щоб NSProxy і інші кореневі класи могли бути використані по черзі з підкласами NSObject.

Так як Objective-C відбувся від Smalltalk із принципом «усе є об'єктом», не дивно, що протоколи, як і класи, теж об'єкти. Ви можете посилати їм повідомлення для проведення інтроспекції. Звичайно ви не робите це самі, замість чого покладаючись на повідомлення, послані до підкласів NSObject:

```
if ([object conformsToProtocol:@protocol(MyDelegateProtocol)])
{
  // Дії з делегованим протоколом
}
```

Річчю, що небагато бентежить у протоколах, є той факт, що вони перевіряються на рівність простим порівнянням імен. Якщо у вас є два протоколи з однаковими іменами, то у вас немає способу вказати в рантаймі, що з них реалізує об'єкт.

Причиною цьому послужив дозвіл перевірки для узгодження за допомогою директиви @protocol(), як продемонстровано вище, у вихідних файлах, у яких немає доступу до опису протоколу; але це може викликати плутанину.

Неформальні протоколи

Дуже частим шаблоном в Objective-C є ідея неформального протоколу, – колекції методів, які клас може реалізовувати, а може не реалізовувати. Її часте використання – для делегування об'єктів. В Java дуже часто для делегатів очікується реалізація інтерфейсу. У такого підходу часто є набір методів, деякі з яких реалізовані як методи без тіла.

В Objective-C є два шляхи для визначення неформальних протоколів. Перший – визначення категорії на базовому класі (звичайно на NSObject), що надає нульові реалізації кожного методу. Це означає, що кожний клас буде відповідати на повідомлення в інтерфейсі, але щось дійсно робити будуть тільки реалізовані методи. Ви повинні помістити у файл вихідного коду, що використовує неформальний протокол, щось наступне:

```
@implementation NSObject (MyInformalProtocol)
- (void) handleSomethingFrom:(id) sender {}
@end
```

Потім ви просто відправляєте повідомлення handleSomethingFrom: делегованому об'єкту. Зверніть увагу, що вам не потрібний розділ @interface для створення поділу між інтерфейсом і реалізацією для категорій; інтерфейс приватний, а вам не потрібно, щоб щось ще викликало цей метод, крім вашого класу, тому не відкривайте інтерфейс. Хоча цей метод простий, він не ідеальний у багатьох випадках, так як приводить до заповнення координуючої таблиці кореневого об'єкта в основному не використовуваним мотлохом (який може привести до зниження продуктивності в рантайм Apple через те, як там реалізовано кешування).

Інший варіант – робити тестування в рантаймі. Якщо ви пошлете об'єкту повідомлення respondsToSelector:, ви можете з'ясувати, чи реалізує він названий метод. Для делегатів, ви можете потім кешувати IMP для методу, і викликати його прямо пізніше. У методі setDelegate: буде таке:

```
handleMethod = NULL;
if([delegate respondsToSelector:@selector(handleSomethingFrom:))
{
```

```
  handleMethod = [delegate methodForSelector:\
    @selector(handleSomethingFrom:)];
}
```

Потім при використанні, робите наступне:

```
// Еквівалентно [delegate handleSomethingFrom:self];
if (NULL != handleMethod
{
  handleMethod(delegate, @selector(handleSomethingFrom:), self);
}
```

Objective-C 2.0 дає третій варіант, що призначений для використання директиви @optional в оголошенні протоколу. Це більша витрата пам'яті, так як вам необхідно робити тести в рантаймі для визначення, чи реалізує об'єкт, зіставлений протоколу, опціональний метод із цього протоколу.

Повторне відправлення

В C++ ви не посилаєте повідомлення, а викликаєте функції-члени. Це важлива відмінність, так як воно позначає, що викликувані методи семантично схожі з викликом функцій. Objective-C вносить інший шар абстракції. Якщо ви посилаєте повідомлення об'єкту Objective-C, що його не розуміє, виникне виключення. Проте, це не робиться самою мовою.

Бібліотека рантайму має механізм усунення несправностей, коли немає методу для селектора. Вона викликає метод, що інтроспектує одержувача для деякої інформації про тип, поміщає виклик в об'єкт NSInvocation, а потім передає його методу -forwardInvocation: даного об'єкта.

Об'єкт NSInvocation інкапсулює одержувача, селектор і аргументи. Ви можете використовувати цю ідею для відправлення повідомлень високого порядку. Розглянемо наступний приклад:

```
[[anArray map] toUppercase];
```

Метод -map, застосований до масиву, повертає об'єкт проксі за допомогою методу forwardInvocation:, реалізованого подібним чином:

```
- (void) forwardInvocation:(NSInvocation*)anInvocation
{
 SEL selector = [anInvocation selector];
 NSMutableArray * mappedArray = [NSMutableArray array];
```

```
 FOREACHI(array, object)
 {
  if([object respondsToSelector:selector])
  {
   [anInvocation invokeWithTarget:object];
   id mapped;
   [anInvocation getReturnValue:&mapped];
   [mappedArray addObject:mapped];
  }
 }
 [anInvocation setReturnValue:mappedArray];
}
```

FOREACHI – макрос із ?toil?, що просто робить кешування IMP на NSEnumerator. Коли ви посилаєте повідомлення -toUppercase до меп проксі, він робить ітерацію по кожному об'єкті масиву; перевіряє, чи відповідає він селектору й, якщо так, викликає метод з аргументами. Значення, що повертається, додається до нового масиву.

Це майже неможливо зробити в C++. Ви можете використовувати шаблон команди, щоб зробити щось схоже, але тільки реалізувавши спочатку ваш власний механізм диспетчеризації.

Типи й покажчики

Objective-C офіційно не дозволяє визначати об'єкти в стеці. Хоча це не зовсім правда, – можливо, визначити об'єкти в стеці, але дуже складно зробити це коректно, так як порушує угоди про організацію керування пам'яттю. У результаті, кожний об'єкт в Objective-C – покажчик. Деякі типи визначені Objective-C; вони визначені в заголовних файлах як типи C.

Три найбільше часто використовувані нові типи в Objective-C – це id, Class і SEL. id – це покажчик на об'єкт Objective-C. Це еквівалентно void* в C, до якого ви можете привести будь-який тип покажчика на об'єкт, а також привести його до будь-якого іншого типу покажчика на об'єкт. Ви можете спробувати відправити яке-небудь повідомлення до id, але ви одержите виключення в рантаймі, якщо така можливість не підтримується.

Class – це покажчик на клас Objective-C. Класи є об'єктами, тому вони також можуть одержувати повідомлення. Ім'я класу є типом, а не змінної. Ідентифікатор NSObject це тип екземпляра NSObject, але він також може бути використаний як одержувач повідомлень. У вас може бути такий клас:

```
[NSObject class];
```

Цей код посилає повідомлення +class класу NSObject, що повертає покажчик до структури Class, що представляє собою сам клас. Це корисно для інтроспекції, як ми побачимо далі.

Третій тип, SEL, являє собою селектор – абстрактне подання ім'я методу. Ви можете створити його під час компіляції за допомогою директиви @selector() або в рантаймі шляхом виклику функції рантайм бібліотеки з рядком C або використовуючи функцію OpenStep NSSelectorFromString(), що дає селектор для рядка Objective-C. Ця техніка дозволяє вам викликати методи по ім'ю. Ви можете зробити це в C, використовуючи щось начебто dlsym(), але це набагато складніше в C++. В Objective-C ви можете зробити таке:

```
[object performSelector:@selector(doSomething)];
```

Що буде еквівалентно наступному:

```
[object doSomething];
```

Ясно, що другий варіант буде небагато швидший, так як перший виконує відправлення двох повідомлень. Пізніше ми більш детально розглянемо те, що ви можете робити із селекторами.

В C++ немає еквівалента типу id, так як об'єкти завжди повинні бути типізовані. В Objective-C у вас є додаткова система типів. Обоє наступного варіанта будуть коректні:

```
id object = @"a string";
NSString *string = @"a string";
```

Насправді, константний рядок – це екземпляр клас NSConstantString, що є дочірнім класом NSString. Присвоювання його до NSString* включає перевірку типів під час компіляції для повідомлень і доступ до публічних змінних екземпляра (які майже ніколи не використовуються в Objective-C). Ви можете порушити цю установку, зробивши наступне:

```
NSArray *array = (NSArray*)string;
```

Якщо ви відправляєте повідомлення масиву, компілятор перевірить, що вони є повідомленнями, які розуміє NSArray. Це не дуже корисно, так як об'єкт є рядком. Якщо ви посилаєте йому повідомлення, що є одночасно реалізаціями NSArray і NSString, це, проте, спрацює. Якщо ж ви відправляєте йому повідомлення, які NSString не реалізує, виникне виключення.

У той час, як робити таке може здатися дивним (а це так і є, так що не робіть так), це підкреслює дуже важливу відмінність між Objective-C і C++. В Objective-C певна семантика типів, у той час як C++ має змінну семантику типів. В C++ тип залежить від типу змінної. Коли ви привласнюєте покажчику на об'єкт в C++ змінну, певну як покажчик на суперклас, два покажчики можуть не мати однакове чисельне значення (це зроблено для того, щоб дозволити множинне спадкування, що не підтримує Objective-C).

Визначення класів

Визначення класів в Objective-C є секції інтерфейсу й реалізації. В C++ є щось схоже, але там це в якімсь ступені змішаний. Інтерфейс в Objective-C тільки визначає частини, які точно повинні бути загальнодоступні. З метою реалізації, він містить у собі приватні змінні екземпляра, так як ви не можете зробити клас суперкласом, поки не знаєте, наскільки він великий. Більше пізні реалізації, як, наприклад, 64-бітний рантайм від Apple, не має цього обмеження.

Інтерфейс об'єктів в Objective-C виглядає в такий спосіб:

```
@interface AnObject : NSObject  {
@private
    int integerivar
@public
    id anotherObject;
}
+ (id) aClassMethod;
- (id) anInstanceMethod:(NSString*)aString with:(id)anObject
@end
```

Перший рядок містить три частини: ідентифікатор AnObject – ім'я нового класу. Ім'я після двокрапки – це об'єкт NSObject (це необов'язково, але майже

кожний об'єкт в Objective-C розширює NSObject). Імена в кутових дужках це протоколи, – це схоже з інтерфейсами в Java – які реалізуються цим класом.

Як і в C++, змінні екземпляра («поля» в C++) можуть мати специфікатори доступу. У відмінності від C++, перед цими специфікаторами ставиться знак «@», щоб уникнути збігів з коректними ідентифікаторами C.

Objective-C не підтримує множинне спадкування, так що існує тільки один суперклас. Тому формат першої частини об'єкта завжди ідентичний формату екземплярів суперкласу. Це використано для статичного визначення, що означає, що зміна змінних екземпляра в одному класі вимагає перекомпіляцію всіх підкласів. У більше нових рантаймах це визначення не потрібно, ціною цьому став більше дорогий доступ до змінних екземпляра. Інший побічний ефект цього рішення те, що він руйнує одну з інших можливостей мови Objective-C:

```
struct _AnObject
{
    @defs(AnObject);
};
```

Директива @defs означає: «Помістити всі поля зазначеного об'єкта в цю структуру», тому структура _AnObject має точно такий же формат у пам'яті, як і екземпляр класу AnObject. Ви можете використовувати це правило, наприклад, для доступу до змінних екземпляра прямо. Часто це використовують, щоб дозволити функції C управляти об'єктом Objective-C прямо, з метою підвищення продуктивності.

Також за допомогою цієї можливості, ви можете створювати об'єкти в стеці, на що я й натякав раніше. Так як в структури й об'єкта однаковий формат розміщення в пам'яті, ви просто створюєте структуру, установлюєте її покажчик на коректний клас, а потім приводите покажчик до покажчика на об'єкт. Потім ви можете використовувати його як об'єкт, хоча ви повинні бути дуже обережні: ніщо не повинне зберігати покажчики до нього за межами області видимості структури.

На відміну від C++, в Objective-C немає приватних або захищених методів. Будь-який метод об'єкта Objective-C може бути викликаний будь-яким іншим

об'єктом. Якщо ви не оголосили метод в інтерфейсі, він неформально приватний. Під час компіляції ви одержите попередження, що об'єкт може не відповісти на це повідомлення, але ви можете як і раніше викликати його.

Інтерфейс схожий з раннім оголошенням в C. Йому так само потрібна реалізація, що використовує @implementation, що не дивно:

```
@implementation AnObject
+ (id) aClassMethod
{
 ...
}
- (id) anInstanceMethod:(NSString*)aString with:(id)anObject
{
 ...
}
@end
```

Зверніть увагу, як зазначені типи параметрів, у дужках. Це повторне використання синтаксису приведення в C, щоб показати, що значення приводяться до цього типу. Насправді, вони можуть і не бути такого типу. Точно такого ж правила застосовні тут по час приведення. Це позначає, що приведення між несумісними типами покажчиків викликають попередження (а не помилку).

4.2 Захист розробленого програмного забезпечення

Дані у програмному забезпеченні я захищаю за допомогою MISTY1. MISTY1 – блоковий алгоритм шифрування, створений для компанії Mitsubishi Electric криптологом Міцуру Мацуї. Назва є абревіатурою Mitsubishi Improved Security Technology. Алгоритм був розроблений в 1995-1996 рр. Відомі також дві модифікації алгоритму MISTY1: MISTY2 і KASUMI

Шифр став переможцем на Європейському конкурсі NESSIE. У результаті аналізу алгоритму експерти зробили вивід, що ніяких серйозних уразливостей даний алгоритм не має (переважно, завдяки вкладеним мережам Фейстеля, що

суттєво утрудняє криптоаналіз). У нього високий запас криптостійкості, алгоритм має високу швидкість шифрування й досить ефективний для апаратної реалізації.

Алгоритм був розроблений на основі теорії «підтвердженої безпеки» проти диференціального й лінійного криптоаналізу. Цей алгоритм був спроектований, щоб протистояти криптоатакам, відомим на момент створення.

З моменту публікації MISTY1 було проведено багато досліджень, щоб оцінити його рівень безпеки.

Диференціальний і неможливий диференціальний криптоаналіз високого порядку ефективно застосовується до блокових шифрів з малим ступенем. Найкращі результати для обох варіантів були отримані для 5-рівневого алгоритму MISTY1 без FL функцій.

Саме FL функції й широкобітні AND/OR операції в сильно утрудняють використання диференціального криптоаналізу, що не заважає проведенню в цьому напрямку всі нових досліджень і досягненню усе більш близьких до розв'язку результатів.

Параметри вихідних даних

MISTY1 – це шифр на основі вкладених мереж Фейстеля з варьюємим числом раундів. Рекомендоване використання 8-раундової версії, але може використовуватися будь-яка кількість раундів, кратне 4-м. Розмір блоку вихідного тексту – 64 біта, розмір ключа – 128 біт.

Для роботи алгоритму також попередньо виконується процедура розширення ключа, яка для 8-мі раундів обчислює 1216 бітів ключової інформації з 128-бітного ключа шифрування.

Структура алгоритму

Для задоволення вимогам конкурсу NESSIE, а також для задоволення завдання мультиплатформеності, в алгоритмі MISTY1 використовувалися наступні методи шифрування:

– Логічні операції.

– Арифметичні операції.

– Операції зрушення.

– Таблиці перестановок.

Як говорилося вище, алгоритм MISTY1 заснований на «вкладених» мережах Фейстеля. Спочатку блок вихідного тексту розбивається на два 32-бітних субблоки, після чого виконується r раундів наступних перетворень[1]:

– У кожному непарному раунді обоє субблоки обробляються операцією FL

– Над оброблюваним субблоком виконується операція FO.

– Результат цих операцій накладається логічною операцією «, що виключає або» (XOR) на неопрацьований субблок.

– Субблоки міняються місцями. Після заключного раунду обоє субблоки ще раз обробляються операцією FL.

Операція FL

Оброблюваний 32-бітний субблок розбивається на два 16-бітних фрагмента, до яких застосовуються операції, де:

– L і R – вхідні значення лівого й правого фрагментів відповідно;

– L' і R' – вихідні значення;

– і – фрагменти j-го підключа i-го раунду для функції FL (процедура розширення ключа докладно описана далі);

– і – побітіві логічні операції «і» і «або» відповідно.

Операція FO

Саме ця функція є вкладеною мережею Фейстеля. Тут, як і раніше, виконується розбивка вхідного значення на два 16-бітних фрагмента, що проходять 3 раунду наступних перетворень:

– На лівий фрагмент операцією XOR накладається фрагмент ключа, де k – номер раунду функції FO.

– Лівий фрагмент обробляється операцією FI.

– На лівий фрагмент накладається операцією XOR значення правого фрагмента.

– Фрагменти міняються місцями.

Після третього раунду операції FO на лівий фрагмент накладається операцією XOR додатковий фрагмент ключа.

Операція FI

Дана операція також представляє собою третій рівень вкладеності мережі Фейстеля. На відміну від двох верхніх рівнів, дана мережа є незбалансованою: оброблюваний 16-бітний фрагмент ділиться на дві частини: 9-бітну ліву й 7-бітну праву. Потім виконуються 3 раунду перетворень, що випливають:

– Ліва частина зазнає обробці S-box.9-бітна частина (в 1-м і 3-м раундах) обробляється таблицею S9, а 7-бітна (в 2-м раунді) – таблицею S7. Дані таблиці описані нижче.

– На ліву частину операцією XOR накладається поточне значення правої частини. При цьому, якщо праворуч 7-бітна частина, вона доповнюється нулями ліворуч, а в 9- бітної частини віддаляються ліворуч два біти.

– У другому раунді на ліву частину операцією XOR накладається фрагмент ключа раунду, а на праву – фрагмент. В інших раундах ці дії не виконуються.

– Ліва й права частини міняються місцями.

Для оптимального розв'язку завдання мультиплатформеності, таблиці S7 і S9 алгоритму MISTY1 можуть бути реалізовані як за допомогою обчислень, так і безпосередньо таблицями.

Розширення ключа

Для 8 раундів алгоритму результатом процедури розширення ключа буде наступний набір ключових значень:

– 20 фрагментів ключа (), кожний з яких має розмір по 16 бітів;

– 32 16-бітних фрагмента ();

– 24 7-бітних фрагмента (при k=4, тобто в 4-м раунді функції FO, операція FI не виконується);

– 24 9-бітних фрагмента.

Виконується дане обчислення в такий спосіб:

1. 128-бітний ключ ділиться на 8 фрагментів … по 16 бітів кожний.

2. Формуються значення: у якості використовується результат обробки значення функцією FI, яка в якості ключа (тобто сукупності необхідних 7- і 9-бітного фрагментів) використовує значення(якщо індекс n фрагмента ключа перевищує 8, то замість нього використовується індекс n-8).

Необхідні фрагменти розширеного ключа «набираються» у міру виконання перетворень із відповідних масивів і згідно з відповідними таблицями

16-бітний фрагмент ділиться на 7-бітний фрагмент і 9-бітний .

Розшифрування

Розшифрування проводиться виконанням тих же операцій, що й при зашифруванні, але з наступними змінами:

– фрагменти розширеного ключа використовуються у зворотній послідовності,

– замість операції FL використовується зворотна їй операція – FLI.

Схеми виконання функції FLI і процедури розшифрування наведено на малюнках 6 і 7 відповідно:

Методи аналізу

Як говорилося на початку розділу, диференціальний і неможливий диференціальний аналізи виявилися ефективні лише до версій шифру з меншою кількістю раундів і без операції FL [2][3]. Проте, на даний момент цей напрямок аналізу, особливе використання слабких ключів, найбільше перспективно, тому що наближене до реальних можливих допущень при використанні алгоритму.

Так само, ученим з Японії був проведений інтегральний аналіз повного алгоритму, використовуючи відкритих текстів зі складністю обчислення, рівної [4].

Лінійний аналіз дав результати тільки для 7-раундової версії шифру, і також без операції FL[5].

Так як MISTY1 створювався, у тому числі, з розрахунку на апаратну реалізацію, має сенс диференціальний аналіз, заснований на використанні атаки по помилках обчислень, що в цьому випадку наближене до реальності.

Висновок

Таким чином, була докладно описана структура алгоритму шифрування MISTY1 і розглянуті методи його аналізу, найбільш прагматичні напрямки дослідження. Далі має бути створення програмної реалізації для більш детального розгляду алгоритму й набір статистичних даних для повного дослідження й пошуку оптимального підходу до аналізу MISTY1.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

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

– Функціональна клавіша перегляд файлів.

– Функціональна клавіша моніторинг системи.

– Функціональна клавіша налаштування ПЗ.

– Функціональна клавіша архівування файлів та каталогів.

– Функціональна клавіша перегляду авторського права.

– Функціональних клавіш першого сканування.

Рисунок 5.1 – Головне вікно програми

На рисунку 5.2 зображено форму авторського права. Було обрано Shareware умову розповсюдження. Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (неповнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми. В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання. Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно.

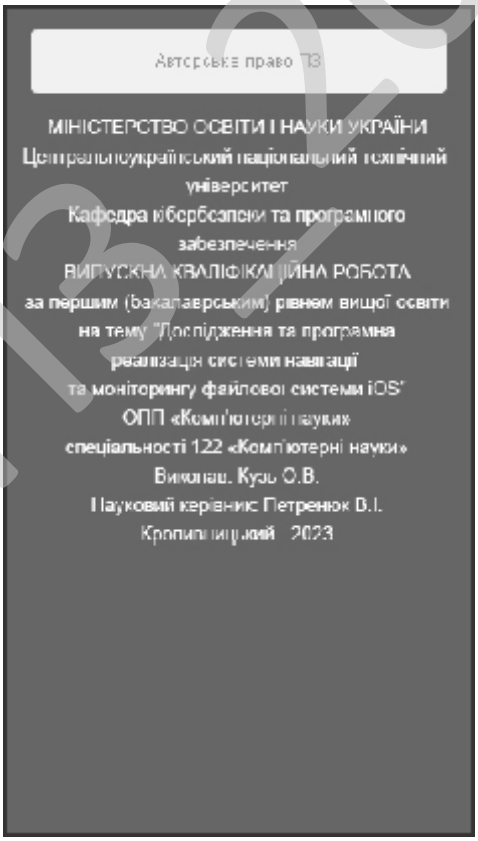

Рисунок 5.2 – Довідка автора

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи навігації та моніторингу файлової системи iOS.

Метою розробки є дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS.

Об'єктом дослідження є процес навігації та моніторингу файлової системи iOS.

Предметом дослідження є методи навігації та моніторингу файлової системи iOS.

Методи дослідження базуються на методах побудови файлових систем, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод навігації та моніторингу файлової системи iOS.

– Розроблено вітчизняний продукт навігації та моніторингу файлової системи iOS, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

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

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В магістерській роботі було проведене дослідження та виконана програмна реалізація системи навігації та моніторингу файлової системи iOS.

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

а) невеликий розмір;

б) невеликі системні потреби;

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

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	99
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

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

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	100000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	55
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

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

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

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

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 70 = 118 \text{ люд/день.}$$

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

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	118	Ф 7.1-7.4
Впровадження	13	Д13
Всього	159	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

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

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

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

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

$$Ч = \frac{159 \cdot 1}{60 - 5} = 2,9 \text{ ставки.}$$

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

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

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

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	7	630	10,5
Монітор	60	7	420	7
Клавіатура	30	7	210	3,5
Маніпулятор «мишка»	30	7	210	3,5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор-маршрутизатор	30	1	30	0,5
Кабельні господарства ЛОМ на 1 м.п.	2,5	160	400	6,67
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	37,33

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

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

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

$$Ф_{др}^{с} = \frac{37 \cdot 3}{1,2} = 92,5 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

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

$$Ч_{ел} = 92,5/(60 \cdot 8) = 0,2 \text{ ставки.}$$

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

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

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

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

Продовження таблиці 7.4

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

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	7870	23610
Продакт-менеджер	0,25	7000	5250
Інженер-програміст	2,9	8500	73950
Інженер - електронщик	0,2	7000	4200
Інженер-системотехнік	0,25	7000	5250
Адміністратор мережі	0,5	7000	10500
Системний програміст	0,1	7000	2100
Дизайнер WEB	0,25	7000	5250
Інженер-верстальник	0,25	7000	5250
Бухгалтер-економіст	0,1	7000	2100
Всього за період розробки	$R_{сп}$ = 5,8	-	$Ф_{роб}$ = 137460

Розрахуємо середньоденну зарплату одного виконавця:

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

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

$$З_{сд} = \frac{137460}{5,8 \cdot 60} = 395 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

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

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

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

$S_{у}$ – питома площа на одне робоче місце, $м^2$;

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

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно $8 м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 29000 = 1858000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 185800 грн.

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

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

де: $Ц_{м}$ – ціна меблів для одного робочого місця, грн.

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

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

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу Інтернет-магазину Компбест за 20.10.23 – джерело https://compbest.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		10947
Системний блок		7347
Процесор	Intel Xeon E3-1240 v5 (4 (8) ядра по 3.5 - 3. GHz), 8 MB Smart Cache	-
Системна плата	HP Z240, 2x PS/2, 1x DVI, 1x DisplayPort, 8 USB 3.0, 2x USB 2.0, 1x Ethernet, 1x COM порт, 4x Audio	-
Відеокарта	nVidia Quadro K620, 2 GB GDDR3, 128-bit	-
Жорсткий диск	256 GB SSD M.2 NVMe, 3,5" SEAGATE TB 7200 rpm SATA-3	-
Оперативна пам’ять	8 GB DDR4	-
DVD-привод	DVD -RW/+RW , LG SATA SuperMulti Bul 22x, SecurDisc, black	-
Корпус	ATX Middle Tower, PSU 350W(FSP Branc ATX-450PNR, 12cm), black, (front bezel black+light silver; body material – 0.6mm) 80mm fan (rear), 4xUSB3.0/AUDIO/MIC, Ai Duct, Tool-less chassis design,Thermall Advantaged Chassis	-
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-E int. 3.5", 1*USB2.0+AUDIO+1394, multi: A Type Cards, black	-

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
інше	Клавіатура, мишка	Подару-нок
Монітор	22" LG 22MP58VQ-P 5 мс IPS 1920x108 250/1000M:1 178/178 D-Sub+HDMI+DVI	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

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

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

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1858000	-	-
2. Передавальні пристрої	185800	-	-
Всього по групі	2043800	5	102190
Група 4			
3. Обчислювальна техніка	199177	-	-
Всього по групі	199177	50	99588,5
4. Нематеріальні активи	100000	10	10000
Група 5, 6			
5. Вимірювальні пристрої	9031	25	2257,75
6. Транспортні засоби	143000	20	28600
7. Господарський інвентар	28000	25	7000
Всього по групі	180031	-	37857,75
Разом	$К_р$ = 2523008		$А_р$ = 249636,25

Примітка: вартість автомобіля Sens (Standard+) взята по даним з автосалону «Кіровоград-Авто», джерело http://kirovograd-avto.ukravto.ua /catalog/tm-9/model-80/description, складає 143000 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

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

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

$$З_о = 395 \cdot 159/99 = 635 \text{ грн.}$$

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

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

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

$$З_д = 635 \cdot 10 \cdot 0{,}01 = 64 \text{ грн.}$$

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

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

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

$$С_{оц} = 0{,}01 \cdot 22(635+64) = 154 \text{ грн.}$$

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

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

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 635 \cdot 15 \cdot 0{,}01 = 95 \text{ грн.}$$

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

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

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

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 1,5 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=206 грн., визначаємо вартість паперу за період розробки:

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

$$З_{М1} = 206 \cdot 1,5 = 309 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 51):

$$З_{М2} = \sum Ц_д , \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 23,6 грн./шт., DVD-R box – 35 грн./шт.

$$З_{М2} = 23,6 \cdot 50+35 = 1215 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

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

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

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

$$З_М = (309 + 1215 + 1702)/99 = 33 \text{ грн.}$$

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

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

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

$$О_п = 635 \cdot 15 \cdot 0,01 = 95 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 99 прим.):

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

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

$$A_м = 249636 \cdot 3/(99 \cdot 12) = 631 \text{ грн.}$$

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

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн
1	2	3
1. Основна зарплата виконавців	$З_о$	635
2. Додаткова зарплата виконавців	$З_д$	64
3. Відрахування на соціальні потреби	$С_{оц}$	154
4. Загальногосподарські витрати	$Г_{осп}$	95
5. Витрати на матеріали	$З_М$	33
6. Освоєння нових операційних систем, мов програмування	$О_п$	95
7. Амортизація основних фондів	$А_м$	631
8. Повна собівартість програмного забезпечення	$С_п$	1707
9. Плановий прибуток	$П_р$	940
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	2647
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	529,4
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	3176,4

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

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

$$C_п = 635+64+154+95+33+95+631 = 1707 \text{ грн.}$$

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

Для даного програмного забезпечення рівень рентабельності складає 50%.

$$П_р = 0{,}01 \cdot P_п \cdot C_п, \quad (7.22)$$

де: $P_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 55 \cdot 1707 = 940 \text{ грн.}$$

7.6 Визначення об'єму капітальних вкладень у споживача програмної продукції

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

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

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	3176
Всього капітальних витрат	–	3176

7.7 Визначення експлуатаційних витрат

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

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	53465	11139
2. Витрати на електроенергію	$З_{ел}$	346	72
3. Витрати на амортизацію	$З_{ам}$	0	794
Всього витрат за рік	$І$	53811	12005

Витрати на обслуговування системи:

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

де: $Т_р$ – кількість годин обслуговування системи на рік, *год.*;

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

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 240 годин на рік до 50 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\,баз} = 240 \cdot 166 \cdot 1{,}1 \cdot 1{,}22 = 53465 \text{ грн},$$

до:

$$З_{р\,нов} = 50 \cdot 166 \cdot 1{,}1 \cdot 1{,}22 = 11139 \text{ грн.}$$

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

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

$$З_{ел\,баз} = 0{,}45 \cdot 240 \cdot 3{,}2 = 346 \text{ грн.}$$

$$З_{ел\,нов} = 0{,}45 \cdot 50 \cdot 3{,}2 = 72 \text{ грн.}$$

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

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	3176	–	794
Всього відрахувань	-	–	3176	–	794

7.8 Визначення економічної ефективності програмної продукції

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

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

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

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

$$Eв = (2647\text{-}1707) \cdot 99\text{-}(0{,}05 \cdot 2043800 + 0{,}5 \cdot 199177 + 0{,}25 \cdot 37031 + $$
$$+0{,}1 \cdot 100000 + 0{,}2 \cdot 143000) \cdot 3/12 = 30651 \; грн.$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

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

де: K_p^* – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{479208}{(2647\text{-}1707)\cdot 99\cdot 12/3} = 1{,}3 \text{ років}.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	99
2. Повна собівартість розробленої програми	Грн.	1707
3. Ціна розробленої програми	Грн.	2647
4. Плановий прибуток від реалізації розробленої програми	Грн.	940
5. Рентабельність програмної продукції	%	55
6. Об’єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	2523008
7. Загальний прибуток від реалізації програмної продукції	Грн.	93060
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	30651
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	1,3
10.Об’єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	3176
11.Величина економічного ефекту у користувача програмної продукції	Грн.	41012
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,1

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

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

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

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

$$E_{сп} = (53811\text{-}12005)\text{-}0{,}25 \cdot 3176 = 41012 \text{ грн.}$$

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

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

$$T_{сп} = \frac{3176}{53811 - 12005} = 0{,}1 \text{ року}.$$

7.9 Висновки

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

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Електронно-обчислювальна машина (ЕОМ) відіграє важливу роль у житті сучасної людини. Кожного дня мільйони людей використовують ЕОМ для пошуку необхідної інформації, спілкуванні у соціальних мережах, перегляду новин, роботи тощо. Багато людей користуються ЕОМ у професійних цілях, оскільки завдяки ЕОМ з'явилося багато нових професій.

Аналізуючи умови працівників ІТ-сфери, на перший погляд, може здатися, що працівники сфери інформаційних технологій не схильні до ризиків на виробництві, та якщо більш глибоко розглянути умови і специфіку праці фахівців сфері ІТ-індустрії, можна виявити ряд факторів які будуть мати негативний сплив на стан охорони праці, та на самого ІТ-фахівця зокрема. Сюди можна віднести як невідповідність освітлення, так і високий рівень шуму, що негативно позначатимуться як на емоційному так і на фізичному стані фахівця, призводитимуть до зниження ефективності праці та виробничих травм. Також, важливим моментом охорони праці ІТ-фахівця є врахування його психологічних можливостей (швидкість реакції, особливості пам'яті та уваги, емоційний стан,тощо). Для того, щоб забезпечити ефективну роботу ІТ-фахівця, потрібно враховувати та максимально компенсувати такі негативні фактои як: надмірне нервово-емоційне навантаження, довготривалі статичні перевантаження, обмежена рухова активність. Всі ці чинники призводить до різноманітних відхилень у стані здоров'я, зокрема до перевтоми, зниження фізичної та розумової працездатності, неврозів, захворювань серцево-судинної системи тощо. Метою даного розділу є огляд конкретних умов праці спеціаліста у сфері ІТ-індустрії. Завданнями для даного розділу є: аналіз умов праці на робочому місці фахівця ІТ-індустрії, розробка конкретних рекомендацій щодо покращення умов

праці фахівців ІТ-індустрії, огляд пожежної безпеки на ІТ-підприємстві та розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ – фахівці.

8.2 Аналіз умов праці на робочому місці ІТ-фахівця

На робочому місці ІТ-фахівця (або програміста) виникають небезпечні та шкідливі для безпечної життєдіяльності фактори:

– підвищений рівень шуму;

– несприятливі мікрокліматичні умови;

– недостатній рівень освітленості;

– шкідливі речовини;

– підвищений рівень електромагнітних випромінювань радіочастот;

– висока напруга електричної мережі;

– статична електрика та інші.

Робота програміста супроводжується також підвищеним ступенем напруженості трудового процесу. При систематичному впливі виробничих факторів, які не відповідають нормативним показникам, зростає рівень професійно зумовленої захворюваності працюючих та можуть виникнути професійні захворювання органів зору, руху, нервової системи. Таким чином, вивчення умов праці на робочому місці програміста є необхідною умовою запобігання негативних наслідків впливу небезпечних та шкідливих факторів. Робоче місце, добре пристосоване до трудової діяльності інженера, правильно і доцільно організоване, щодо простору, форми, розміру забезпечує йому зручне положення при роботі і високу продуктивність праці при найменшому фізичному і психічному напруженні.

Нормування параметрів проводиться в залежності від періоду року та категорії важкості виконуваних робіт. Для постійних робочих місць, якими є робочі місця ІТ-фахівців, встановлені оптимальні параметри мікроклімату, а за неможливості їх дотримання використовують допустимі параметри. Робота ІТ-

фахівця за важкістю відноситься до Іа (роботи, що виконуються сидячи і не потребують фізичного напруження) та Іб (роботи, що виконуються сидячи, стоячи або пов'язані з ходінням та супроводжуються деяким фізичним напруженням) категорій. В таблиці 8.1. наведені оптимальні параметри мікроклімату в приміщеннях.

Таблиця 8.1 – Параметри мікроклімату для приміщень з ПК

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	22...24°С; 40... 60%; до 0,1 м/с
Теплий	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	23...25 °С 40...60% 0,1...0,2 м/с

Виміряні за допомогою приладів температура та вологість у приміщеннях праці ІТ-фахівців повинні відповідати зазначеним у таблиці для теплого періоду року. Слід зазначити, що для нормалізації параметрів мікроклімату слід використовувати у приміщеннях кондиціювання повітря, або забезпечити подачу свіжого повітря системами вентиляції. Норми подачі свіжого повітря наведені у таблиці 8.2.

Таблиця 8.2 – Норми подачі свіжого повітря в приміщення

Характеристика приміщення	Об'ємна витрата свіжого повітря, що подається в приміщення, $м^3$ на одну людину в годину
Об'єм до 20 $м^3$ на людину 20... 40 $м^3$ на людину Більше 40 $м^3$на людину	Не менше 30 Не менше 20 Може біти використана природна вентиляція

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення його привабливості, позитивно впливає на продуктивність праці. Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою. У службових приміщеннях, у яких виконується одноманітна розумова робота, що вимагає значної нервової напруги і великого зосередження, забарвлення повинно бути спокійних тонів – малонасичені відтінки холодного зеленого або блакитного кольорів.

При розробці оптимальних умов праці програміста необхідно враховувати освітленість. Раціональне освітлення робочого місця є одним з найважливіших факторів, що впливають на ефективність трудової діяльності людини, що попереджають травматизм і професійні захворювання. Правильно організоване освітлення створює сприятливі умови праці, підвищує працездатність і продуктивність праці. Освітлення на робочому місці програміста повинно бути таким, щоб працівник міг без напруги зору виконувати свою роботу. Стомлюваність органів зору залежить від ряду причин: недостатність освітленості; надмірна освітленість; неправильний напрям світла. Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань. [2]

8.3 Пропозиції щодо підвищення працездатності ІТ-фахівців

Поява та впровадження нових інформаційно-комунікаційних технологій зумовлює необхідність подальшого вдосконалення охорони праці фахівців ІТ-індустрії. Все це потребує розробки нових нормативно-правових актів з регламентації праці та відпочинку фахівців ІТ-індустрії і стандартів підприємств,

центрів комп'ютерної техніки, центрів інформаційних технологій, сучасних комп'ютерних класів. Для підвищення розумової працездатності то зорової роботи повинна здійснюватися ергономічна оптимізація в рамках системи «оператор-термінал», яка сприятиме результативній фізичній та інтелектуальний працездатності і відновленню психосоматичного здоров'я фахівців ІТ-індустрії. Всі наведені заходи щодо вдосконалення охорони праці фахівців ІТ-індустрії повинні контролюватися службою охорони праці та комісією з охорони праці підприємства. Особливе значення у соціальному захисті цієї категорії працівників належить прийняття комплексного договору, який може забезпечити фахівців додатковими пільгами та компенсаціями.

Для більшого розуміння, пропозиції щодо підвищення працездатності ІТ-фахівців, розіб'ємо на декілька категорій:

1 Середовище і розпорядок праці. Для мінімізації негативних ефектів, що пов'язані з перевтомленням ІТ-фахівців, потрібно чітко прописати і реалізувати графік періодів праці-відпочинку, щоб фахівець міг можливість переключити увагу, дати можливість відпочити очам, мозку, елементарно, встати розім'яти ноги. Також потрібно зробити максимально комфортними умови мікроклімату у офісному приміщенні, де працюють ІТ-фахівці. Мається на увазі встановлення і експлуатація, коли виникає необхідність, кондиціонерів, опалення, та системи вентиляції, задля попередження перегрівання, переохолодження ІТ-фахівців, і подальшої неможливості ними виконувати свої функції. Також, за можливості, нами пропонується введення практики віддаленої праці ІТ-фахівцями, якщо роботодавець не може забезпечити оптимальні і безпечні умови в офісному приміщенні, або якщо фахівця вони не влаштовують із певних причин.

2 Фізичні і психоемоційні чинники. Першим і найважливішим чинником, що впливає на працездатність ІТ-фахівців є робоче місце, і саме тому, роботодавець має забезпечити максимальний його комфорт і безпеку. Гарантією цих факторів може слугувати сертифікація меблів, що використовуються на підприємстві ІТ-галузі. Тому нами пропонується закупівля тільки меблів , які

пошли сертифікацію на відповідність. Під психоемоційними чинниками ми розуміємо гарне самопочуття фахівців, позитивний настій, гарний психологічний клімат у колективі, тощо. Задля того, щоб психоемоційні чинники мали максимально позитивний ефект, керівництву слід поводити заходи, які сприятимуть укріпленню і покращенню міжособистісних стосунків у колективі, таких як психологічні тренінги, тімбілдінг, спортивні змагання і естафети. Також, сюди можна віднести розробку і впровадження системи мотивації працівників, як фінансової, так моральної і адміністративної.

8.4 Розрахунок занулення

Мета розрахунку – визначити умови, за яких швидко вимикається пошкоджена електроустановка, тобто визначити поперечний переріз нульового дроту, який забезпечить протікання струму однофазного короткого замикання, достатнього для спрацювання максимального захисту.

Розрахунок заземлення нейтралі і повторного заземлення нульового робочого провідника, що забезпечують безпеку дотику до занулених пристроїв,

Схема електропостачання зануленої електроустановки представлена на рисунку 8.1, де Тр U_1/U_2 – понижуючий трансформатор масляний, схема з'єднання обмоток – зірка-зірка;

Електромережа виконана як двохпровідна мережа, що складається з фазного дроту і нульового захисного провідників.

$$L_1=45 \text{ м}; L_2=450 \text{ м}; P_1=2550 \text{ Вт}.$$

Матеріал жили – мідь.

Постановка задачі занулення електроустановки: визначення такого перетину нульового захисного провідника при якому струм короткого замикання $I_{кз}$ в задане число раз До перевищить номінальний струм спрацьовування апарату захисту $I_{ном}$, Що забезпечить відключення пошкодженого споживача.

1) Вибір типу автоматичного вимикача.

1а) Визначення струму, що споживає електроустановка потужністю P_1 = 2550 Вт:

$$I_{1н} = P_1/U_ф = 2550/220 = 11{,}59 \text{ А} \quad (8.1)$$

де $U_ф$ – фазна напруга (220 В);

1б) Визначення розрахункової величини струму спрацьовування захисного апарату:

$$I_{расч} = (К_п/К_т)I_{1н} = (3/2.5)11{,}59 = 13{,}909 \text{ А}. \quad (8.2)$$

де $К_п = 3$ – коефіцієнт кратності пускового струму;

$К_т = 2.5$ – коефіцієнт тяжкості пуску електроустановки (залежить від часу пуску: t = 5 сек., пуск легкий).

1в) Вибір типу автоматичного вимикача і визначення величини струму спрацьовування апарату захисту:

$I_{ном}$ = 16 А; тип автоматичного вимикача АЕ2026.

2) Визначення струму короткого замикання фази на корпус електроустановки:

$$I_{кз} = U_ф/ Z_{пфн} \quad (8.3)$$

$Z_{пфн}$ – опір петлі фаза-нуль.

2а) Перетин фазного дроту визначається залежно від допустимого тривалого струму, способу прокладки дротів і матеріалу дротів:

Для знаходження перетину дроту, визначимо діаметр дроту d. Де d визначають по значенню струму I [А], і допустимої густини струму J [А/мм^2]. Табличне значення: J=5 [А/мм^2].

$$d=16/5=3.2 \text{ мм}^2$$

8.5 Висновки до розділу

Повсякденний контроль за дотриманням, виконанням правил пожежної безпеки по експлуатації електроприладів, кондиціонерів, вентиляційних систем, систем опалення та застосування заходів до усунення порушень, що можуть призвести до виникнення пожежі проводиться відповідальною особою. Це є основною запорукою пожежної безпеки в приміщеннях установи.

Враження електрострумом високої напруги є фактором короткочасної дії, що відрізняється не постійним погіршенням здоров'я (за винятком фізичного враження нервової системи), а разовим враженням, що приводить до тимчасового шоку чи смерті.

Для зменшення кількості випадків смертельного ураження струмом розроблено обов'язкові заходи. До цих заходів належать технічні захисні конструкції., та правила використання приладів, що є під напругою, використання заземлень корпусів обладнання та занулення струмопровідної мережі.

Отже в даному розділі магістерської роботи були визначені можливі дії установи в якій працюють програмісти, зайняті розробкою вказаного програмного забезпечення, для попередження небезпечних ситуацій відносно найбільш суттєвих факторів ризику.

Список використаних джерел інформації

1. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

2. Державні будівельні норми України: Природне і штучне освітлення ДБН В.2.5-28:2018. – Режим доступу до ресурсу: https://goo.su/9AkQ

3. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення 19.09.22).

4. Визначення категорій приміщень, будинків, установок за вибухопожежною та пожежною небезпекою ДСТУ Б В.1.1-36:2016. Режим доступу до ресурсу: https://dbn.co.ua/load/normativy/dstu/dstu _b_v_1_1_36/5-1-0-1759

5. Державні будівельні норми України: ДБН В.2.5-28:2018. – Режим доступу до ресурсу: https://goo.su/9AkQ

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи навігації та моніторингу файлової системи iOS.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів навігації та моніторингу файлової системи iOS.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем навігації та моніторингу файлової системи iOS.

– Досліджена система навігації та моніторингу файлової системи iOS.

– На основі отриманих результатів досліджень створена програмна реалізація системи навігації та моніторингу файлової системи iOS.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання навігації та моніторингу файлової системи iOS.

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

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

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

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм MISTY1.

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

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 41012 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,1 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кузь О.В. Дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2023.

2. Priscila Heller. Automating Workflows with GitHub Actions. Packt Publishing. 2021. 216 p.

3. JJ Geewax. API Design Patterns. Manning Publications Co. 2021. 481 p.

4. Prateek Prasad. App Design Apprentice. Razeware LLC. 2020. 272 p.

5. Dawn Griffiths, David Griffiths. Head First Android Development. O'Reilly Media, Inc. 2021. 1414 p.

6. Nathan Metzler. Kotlin Programming for Beginners. Independently published. 2021. 158 p.

7. Aaron Torres. Go Programming Cookbook Second Edition. Packt Publishing Ltd. 2019. 427 p.

8. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В.Ф., 2019. – 156 с.

9. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

10. Knuth D. The Art of Computer Programming: Vol. 3: Sorting and Searching 2nd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 800 p.

11. Knuth D. Art of Computer Programming, Vol. 2: Seminumerical Algorithms 3rd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 672 p.

12. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.

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

14. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.

15. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.

16. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143

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

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

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

20. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable

Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

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

22. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

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

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

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

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

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

28. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P.701-706.

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

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

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

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

33. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А. «Дослідження нормативної документації та стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». VI міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології", м. Кропивницький. 20-21 квітня 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 35-36.

34. Смірнов, О.А., Усік П.С., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю. «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв’язку». Проблеми телекомунікацій. № 1(26). С. 83-96. 2020.

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

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

37. Смірнов О.А., Смірнов С.А., Коваленко О.В., Коваленко А.С. Технологія тестування DOM XSS уразливості. Науково-практичний журнал кібер безпеки (SPCSJ) № 1. [Електронний журнал]. Грузія. Тбілісі: SCSA – 2017.

38. Смірнов О.А., Лисенко І.А. Інформаційна технологія проектування тестових наборів з урахуванням вимог до програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 4 (44). – Полтава: ПолтНТУ. – 2017. – С. 112-115.

39. Смірнов О.А., Смірнов С.А., Рябой Д.К., Рябая О.В. Модель вузла комутації з відносними пріоритетами, резервуванням ресурсів і обліком реальної надійності обслуговуючих приладів .Збірник тез всеукраїнської науково-практичної інтернет-конференції «Автоматика та комп’ютерно-інтегровані технології у промисловості, телекомунікаціях, енергетиці та транспорті». м. Кропивницький. 16-17 листопада 2017 р. – Кропивницький: ЦНТУ. – 2017. – С. 198-199.

40. Смірнов О.А., Коваленко О.В. Використання псевдобулевих методів бівалентного програмування для управління ризиками розробки програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ. – 2016. – С. 98-103.

41. Смірнов О.А., Лисенко І.А. Формалізація процесу проектування тестових наборів. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 3 (48). – Харків: ХУПС. – 2016. – С.96-100.

42. Смірнов О.А., Лисенко І.А. Удосконалення методу перевірки коректності таблиць рішень для подання тестових наборів. Збірник наукових праць "Системи обробки інформації". – Випуск 8 (145). – Х.: ХУПС – 2016. – С. 77-80.

43. Смірнов О.А., Лисенко І.А. Розробка впорядкованих каскадних таблиць рішень із використанням матриць слідування. Збірник наукових праць "Системи обробки інформації". – Випуск 6 (143). – Х.: ХУПС – 2016. – С. 216-220.

44. Смірнов О.А., Коваленко О.В., Якименко Н.М., Доренський О.П. Метод кількісної оцінки ризиків розроблення програмного забезпечення. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. – 2016. – С. 128-133.

45. Смірнов О.А., Коваленко О.В., Якименко Н.М., Доренський О.П. Метод якісного аналізу ризиків розроблення програмного забезпечення. Наука і техніка Збройних Сил України. – Випуск 2(23). – Харків: ХУПС. – 2016. – С. 150-158.

46. Смірнов О.А., Коваленко О.В., Якименко Н.М., Доренський О.П. Проблеми аналізу та оцінки ризиків інформаційної діяльності. Збірник наукових праць "Системи обробки інформації". – Випуск 3 (140). – Х.: ХУПС – 2016. – С. 40-42.

47. Смірнов О.А., Коваленко А.С., Коваленко О.В., Доренський О.П. Удосконалення методу технічного обслуговування об’єктів інтегрованої інформаційної системи. Системи озброєння і військова техніка. – Випуск 2(46) – Х.: ХУПС – 2016. – С. 103-107.

48. Smirnov A.A., Kovalenko A.V. Kovalenko A.S. Dorensky A.P. Information model and its element for displaying information on technical condition of

objects of integrated information system. International Journal of Computational Engineering Research (IJCER). – Volume 6, Issue 1. – India. Delhi. – 2016. – P. 21-27.

49. Смірнов О.А., Євсеєв С.П., Король О.Г., Коваленко О.В., Коваленко А.С., Смірнов С.А. Архітектура мікропроцесорів та компонентів ЕОМ. Навчальний посібник – Кіровоград: Вид. Лисенко В.Ф., 2015. – 550 с.

50. Смірнов О.А., Коваленко О.В., Мелешко Є.В., Константинова Л.В., Кожанова А.С. Інженерія програмного забезпечення. Навчальний посібник для студентів вищих навчальних закладів напрямів підготовки 8.050102 «Комп'ютерна інженерія». За ред. О.А. Смірнова Гриф "Навчальний посібник" надано у відповідності з листом Міністерства освіти і науки України від 18.03.2013 року № 1/11-5584. – Кіровоград: КНТУ 2013. – 409с.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи навігації та моніторингу файлової системи iOS.

2 Підстава для розробки

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

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

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

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи навігації та моніторингу файлової системи iOS;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Objective-C.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Економічні вимоги

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

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

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинні бути розглянуті пропозиції щодо підвищення працездатності ІТ-фахівців.

9 Перелік документів, що розробляються

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

10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

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

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 11.12.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

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

_____ Петренюк В.І.

Дослідження та програмна реалізація системи навігації та моніторингу файлової системи iOS

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 44

Літера: РП

Кропивницький – 2023 року

AFCDevice.m – клас-оболонка для класу AFCInterface

```
	/*

	AFCDevice.m
	iPhoneConnection

	 Copyright (c) Литовченко Анна Юріївна, 2014

	 */

// AFCDevice є класом-оболонкою для класу AFCInterface, забезпечуючи розширену функціональність, використовуючи більше Cocoa-дружні класи, такі як NSData.

#import "AFCDevice.h"

#define kMediaAFC @"com.apple.afc"
#define kRootAFC @"com.apple.afc2"

@interface AFCDevice (Private)

-(BOOL)initializeDevice:(AFCDeviceRef *)dev;
-(struct afc_connection *)openDevice:(AFCDeviceRef *)dev withService:(NSString *)service;

@end

@implementation AFCDevice


-(id)initWithRef:(AFCDeviceRef *)dev {

	if (self = [super init]) {

		// По-перше, ініціалізувався пристрій структурою am_device, потім відкрили підключення до нього.

		if ([self initializeDevice:dev]) {
			struct afc_connection *connection = [self openDevice:dev withService:kMediaAFC];

			if (connection) {

				deviceInterface = [[AFCInterface alloc] initWithAFCConnection:connection];

				// Підписатися на центр повідомлень за замовчуванням для повідомлень про відключення.

				[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(disconnected:)

				 name:@"AFC_DeviceWasDisconnected" object:nil];

			} else {
				[self release];
				return nil;
			}
		} else {
			[self release];
			return nil;
		}

	}
```

```
      return self;
}

-(void)dealloc {
      [[NSNotificationCenter defaultCenter] removeObserver:self];
      [deviceInterface release];
      [self setDelegate:nil];
      [super dealloc];
}

#pragma mark -
#pragma mark File Reading

-(NSData *)contentsOfFileAtPath:(NSString *)path {

      // Читає весь файл в об'єкт NSData перед його поверненням.

    if ([deviceInterface isFileAtPath:path]) {

        NSMutableData *data = [[NSMutableData alloc] init];

            // Відкрити файл у режимі 2 (тільки читання)
        unsigned long long rAFC = [deviceInterface openFileAtPath:path
withMode:2];
        unsigned int bufferSize = 256 * 1024; // частини по 256Kb
        unsigned long offset = 0;

        if (rAFC != 0) {

            unsigned int size = bufferSize;
            NSData *chunkData = [deviceInterface readFromFile:rAFC size:&size
offset:offset];

            while (size > 0) {

                [data appendData:chunkData];
                offset += size;
                chunkData = [deviceInterface readFromFile:rAFC size:&size
offset:offset];

            }

            // Переконайтеся, що файл не буде закрито.
            [deviceInterface closeFile:rAFC];
        }


        return [data autorelease];

    } else {

        return nil;

    }

}


-(NSData *)chunkOfFileAtPath:(NSString *)path range:(NSRange)range {

      // Читає частини файлу в об'єкт NSData перед його поверненням.


      if ([deviceInterface isFileAtPath:path]) {

        NSMutableData *data = [[NSMutableData alloc] init];

            // Відкрити файл у режимі 2 (тільки читання)
```

```
        unsigned long long rAFC = [deviceInterface openFileAtPath:path
withMode:2];
        unsigned int bufferSize = 256 * 1024; // 256Kb
        unsigned int bytesToRead = range.length;

            unsigned long offset = range.location;

        if (rAFC != 0) {

                  unsigned int size = bufferSize;

                  if (size > bytesToRead) {
                        size = bytesToRead;
                  }

            NSData *chunkData = [deviceInterface readFromFile:rAFC size:&size
offset:offset];

            while (size > 0) {

                        bytesToRead -= size;

                [data appendData:chunkData];
                offset += size;

                        if (size > bytesToRead) {
                              size = bytesToRead;
                        }

                chunkData = [deviceInterface readFromFile:rAFC size:&size
offset:offset];

            }

                  // Переконайтеся, що файл не буде закрито.
            [deviceInterface closeFile:rAFC];
        }

        return [data autorelease];

    } else {

        return nil;

    }
}


#pragma mark -
#pragma mark File Writing

-(void)createFileAtPath:(NSString *)path withData:(NSData *)data {

      if ([deviceInterface isDirectoryAtPath:[path
stringByDeletingLastPathComponent]]) {
            // Ми можемо лише створити файл, якщо батьківський каталог існує

            unsigned long long rAFC = [deviceInterface openFileAtPath:path
withMode:3];
            unsigned int bufferSize = 256 * 1024; // 256Kb
            unsigned int bytesToWrite = 0;
            unsigned int offset = 0;

            if (rAFC != 0) {

                  if ([data length] < bufferSize) {
                        bufferSize = [data length];
                  }
```

```
                bytesToWrite = [data length];

                while (bytesToWrite > 0) {

                        NSData *chunk = [data
subdataWithRange:NSMakeRange(offset, bufferSize)];

                        // Записуємо частинами

                        [deviceInterface writeToFile:rAFC data:[chunk bytes]
size:bufferSize offset:offset];

                        offset += bufferSize;
                        bytesToWrite -= bufferSize;

                        if (bytesToWrite < bufferSize) {
                              bufferSize = bytesToWrite;
                        }

                }

                // Готово!

                [deviceInterface closeFile:rAFC];

            }
      }
}



#pragma mark -
#pragma mark Filesystem

-(void)deleteFileAtPath:(NSString *)path {

      // Видалити файл (не папку) у даному шляху.

      if ([deviceInterface isFileAtPath:path]) {
            [deviceInterface removePath:path];
      }
}


-(NSArray *)listOfFilesAtPath:(NSString *)path {

      // Повертає список файлів (не папок) у даному шляху, який повинен бути
каталог. Перевіряє, щоб переконатися, що кожен повернутий елемент насправді
файл.

    NSMutableArray *files = [[NSMutableArray alloc] init];

    NSArray *contents = [deviceInterface listFilesInPath:path];

    NSEnumerator *e = [contents objectEnumerator];
    NSString *fileName;

    while (fileName = [e nextObject]) {

        if ([fileName isEqualToString:@""] || [fileName isEqualToString:@"."] ||
[fileName isEqualToString:@".."]) {
          //
        } else if ([deviceInterface isFileAtPath:[path
stringByAppendingPathComponent:fileName]]) {
            [files addObject:fileName];
        }
    }

    return [files autorelease];
```

```
}

-(NSArray *)listOfFoldersAtPath:(NSString *)path {

      // Повертає список папок (не файлів) в даному шляху, який повинен бути
каталог. Перевіряє, щоб переконатися, що кожен повернутий елемент фактично
папка.

    NSMutableArray *folders = [[NSMutableArray alloc] init];

    NSArray *contents = [deviceInterface listFilesInPath:path];

    NSEnumerator *e = [contents objectEnumerator];
    NSString *fileName;

    while (fileName = [e nextObject]) {

        if ([fileName isEqualToString:@""] || [fileName isEqualToString:@"."] ||
[fileName isEqualToString:@".."]) {
            //
        } else if ([deviceInterface isDirectoryAtPath:[path
stringByAppendingPathComponent:fileName]]) {
            [folders addObject:fileName];
        }
    }

    return [folders autorelease];

}

-(AFCInterface *)deviceInterface {
      return deviceInterface;
}

-(void)setDelegate:(id)del {
      [delegate release];
      delegate = [del retain];
}

-(id)delegate {
      return delegate;
}


-(void)setDeviceRef:(AFCDeviceRef *)dev {
      if ([self initializeDevice:dev]) {
            struct afc_connection *connection = [self openDevice:dev
withService:kMediaAFC];

            if (connection) {
                  [deviceInterface release];
                  deviceInterface = [[AFCInterface alloc]
initWithAFCConnection:connection];
            }
      }
}


-(AFCDeviceRef *)device {
      return device;
}

#pragma mark -
#pragma mark Private Methods


-(void)disconnected:(NSNotification *)notification {

      unsigned int disconnectedDev = [[notification object] unsignedIntValue];
```

```
      if (disconnectedDev == [device device]->device_id) {
            //NSLog(@"Був відключений!");

            if ([delegate respondsToSelector:@selector(deviceWasDisconnected:)])
{
                  [delegate performSelector:@selector(deviceWasDisconnected:)
withObject:self];
            }

      }


}


-(BOOL)initializeDevice:(AFCDeviceRef *)dev {

      // Підключення і початок сесії на пристрої.

    int ret = AMDeviceConnect([dev device]);
    if (ret != 0) {
            [NSException raise:@"AFCDevice" format:@"AMDeviceConnect з помилкою
%d", ret];
            return NO;
    }
    if (!AMDeviceIsPaired([dev device])) {
            [NSException raise:@"AFCDevice" format:@"Пристрій підключити не
вдалося"];
            return NO;
    }
    ret = AMDeviceValidatePairing([dev device]);
    if (ret != 0) {
            [NSException raise:@"AFCDevice" format:@"AMDeviceValidatePairing з
помилкою %d", ret];
            return NO;
    }
    ret = AMDeviceStartSession([dev device]);
    if (ret != 0) {
            [NSException raise:@"AFCDevice" format:@"AMDeviceStartSession з
помилкою %d", ret];
            return NO;
    }

      [device release];
      device = [dev retain];
    return YES;
}


-(struct afc_connection *)openDevice:(AFCDeviceRef *)dev withService:(NSString
*)service {

      // Відкрите з'єднання з пристроєм. Значення служби визначає, які служби,
щоб відкрити - @ «com.apple.afc»відкриває служб інформації (фотографії, музика і
т.д.), а @ «com.apple.afc2»відкриває кореневої сервіс, що містить додатки
пристрою, і т.д. і т.п. . Прим com.apple.afc2 запит в Jailbroken iPod/iPhone.


    if (dev == nil) {
            return nil;
    }

    struct afc_connection *hAFC;

    int ret = AMDeviceStartService([dev device], (CFStringRef)service, &hAFC,
NULL);
    if (ret != 0) {
```

```
            [NSException raise:@"AFCDevice" format:@"AMDeviceStartService з
помилкою %d", ret];
            return nil;
    }
    if (hAFC == nil) {  // контрольна перевірка
            [NSException raise:@"AFCDevice" format:@"AMDeviceStartService не
ініціалізувати з’єднання"];
            return nil;
    }
    ret = AFCConnectionOpen(hAFC, 0, &hAFC);
    if (ret != 0) {
            [NSException raise:@"AFCDevice" format:@"AFCConnectionOpen з
помилкою %d", ret];
            return nil;
    }
    return hAFC;
}


@end
```

AFCDevice.h – бібліотека для файлу AFCDevice.m

```
	/*

	AFCDevice.h
	iPhoneConnection

	 Copyright (c) Литовченко Анна Юріївна, 2014

	 */

#import <Cocoa/Cocoa.h>
#import "MobileDevice.h"
#import "AFCInterface.h"
#import "AFCDeviceRef.h"

@interface AFCDevice : NSObject {

	AFCInterface *deviceInterface;
	AFCDeviceRef *device;

	id delegate;
}

-(id)initWithRef:(AFCDeviceRef *)dev;

-(NSData *)contentsOfFileAtPath:(NSString *)path;
-(NSData *)chunkOfFileAtPath:(NSString *)path range:(NSRange)range;


-(NSArray *)listOfFilesAtPath:(NSString *)path;
-(NSArray *)listOfFoldersAtPath:(NSString *)path;

-(void)createFileAtPath:(NSString *)path withData:(NSData *)data;
-(void)deleteFileAtPath:(NSString *)path;

-(AFCInterface *)deviceInterface;

-(void)setDeviceRef:(AFCDeviceRef *)dev;
-(AFCDeviceRef *)device;


-(void)setDelegate:(id)del;
-(id)delegate;

@end
```

```
/* ----------------------------------------------------------------------------
 *   MobileDevice.h - інтерфейс до MobileDevice.framework
*
 * ---------------------------------------------------------------------------- */

#ifndef MOBILEDEVICE_H
#define MOBILEDEVICE_H

#ifdef __cplusplus
extern "C" {
#endif

#if defined(WIN32)
#include <CoreFoundation.h>
typedef unsigned int mach_error_t;
#elif defined(__APPLE__)
#include <CoreFoundation/CoreFoundation.h>
#include <mach/error.h>
#endif

/* Коди помилок*/
#define MDERR_APPLE_MOBILE  (err_system(0x3a))
#define MDERR_IPHONE        (err_sub(0))

/* Apple Mobile (AM*) помилки  */
#define MDERR_OK                ERR_SUCCESS
#define MDERR_SYSCALL           (ERR_MOBILE_DEVICE | 0x01)
#define MDERR_OUT_OF_MEMORY     (ERR_MOBILE_DEVICE | 0x03)
#define MDERR_QUERY_FAILED      (ERR_MOBILE_DEVICE | 0x04)
#define MDERR_INVALID_ARGUMENT  (ERR_MOBILE_DEVICE | 0x0b)
#define MDERR_DICT_NOT_LOADED   (ERR_MOBILE_DEVICE | 0x25)

/* Apple File Connection (AFC*) помилки  */
#define MDERR_AFC_OUT_OF_MEMORY 0x03

/* USBMux помилки  */
#define MDERR_USBMUX_ARG_NULL   0x16
#define MDERR_USBMUX_FAILED     0xffffffff

/* Повідомлення передаються на пристрій повідомлення зворотні виклики:
передається як частина am_device_notification_callback_info.*/
#define ADNCI_MSG_CONNECTED     1
#define ADNCI_MSG_DISCONNECTED  2
#define ADNCI_MSG_UNKNOWN       3

#define AMD_IPHONE_PRODUCT_ID   0x1290
#define AMD_IPHONE_SERIAL       "3391002d9c804d105e2c8c7d94fc35b6f3d214a3"

/* сервіси, які знаходяться на  /System/Library/Lockdown/Services.plist */
#define AMSVC_AFC                   CFSTR("com.apple.afc")
#define AMSVC_BACKUP                CFSTR("com.apple.mobilebackup")
#define AMSVC_CRASH_REPORT_COPY     CFSTR("com.apple.crashreportcopy")
#define AMSVC_DEBUG_IMAGE_MOUNT     CFSTR("com.apple.mobile.debug_image_mount")
#define AMSVC_NOTIFICATION_PROXY    CFSTR("com.apple.mobile.notification_proxy")
#define AMSVC_PURPLE_TEST           CFSTR("com.apple.purpletestr")
#define AMSVC_SOFTWARE_UPDATE       CFSTR("com.apple.mobile.software_update")
#define AMSVC_SYNC                  CFSTR("com.apple.mobilesync")
#define AMSVC_SCREENSHOT            CFSTR("com.apple.screenshotr")
#define AMSVC_SYSLOG_RELAY          CFSTR("com.apple.syslog_relay")
#define AMSVC_SYSTEM_PROFILER       CFSTR("com.apple.mobile.system_profiler")

typedef unsigned int afc_error_t;
typedef unsigned int usbmux_error_t;

struct am_recovery_device;
```

```
struct am_device_notification_callback_info {
    struct am_device *dev;  /* 0    пристрій */
    unsigned int msg;       /* 4    один з ADNCI_MSG_* */
} __attribute__ ((packed));

/* Тип пристрою відновлення функції зворотного виклику повідомлення.
 * Конвертація в правильний тип. */
typedef void (*am_restore_device_notification_callback)(struct
    am_recovery_device *);

/* Це CoreFoundation об'єкт класу AMRecoveryModeDevice. */
struct am_recovery_device {
    unsigned char unknown0[8];                              /* 0 */
    am_restore_device_notification_callback callback;       /* 8 */
    void *user_info;                                        /* 12 */
    unsigned char unknown1[12];                             /* 16 */
    unsigned int readwrite_pipe;                            /* 28 */
    unsigned char read_pipe;                                /* 32 */
    unsigned char write_ctrl_pipe;                          /* 33 */
    unsigned char read_unknown_pipe;                        /* 34 */
    unsigned char write_file_pipe;                          /* 35 */
    unsigned char write_input_pipe;                         /* 36 */
} __attribute__ ((packed));

/* CoreFoundation об'єкт класу AMRestoreModeDevice. */
struct am_restore_device {
    unsigned char unknown[32];
    int port;
} __attribute__ ((packed));

/* Тип функції зворотного виклику, повідомлення пристрою.*/
typedef void(*am_device_notification_callback)(struct
    am_device_notification_callback_info *, void* arg);

/ * Тип _AMDDeviceAttached функції.
 * Конвертація в правильний тип. */
typedef void *amd_device_attached_callback;

/* Тип пристрою відновлення функції зворотного виклику повідомлення.
 * Конвертація в правильний тип. */

      //typedef void (*am_restore_device_notification_callback)(struct
am_recovery_device *);

struct am_device {
    unsigned char unknown0[16]; /* 0 - пустий */
    unsigned int device_id;     /* 16 */
    unsigned int product_id;    /* 20 - перетворюється до AMD_IPHONE_PRODUCT_ID
*/
    char *serial;               /* 24 - перетворюється до AMD_IPHONE_SERIAL */
    unsigned int unknown1;      /* 28 */
    unsigned char unknown2[4];  /* 32 */
    unsigned int lockdown_conn; /* 36 */
    unsigned char unknown3[8];  /* 40 */
} __attribute__ ((packed));

struct am_device_notification {
    unsigned int unknown0;                      /* 0 */
    unsigned int unknown1;                      /* 4 */
    unsigned int unknown2;                      /* 8 */
    am_device_notification_callback callback;   /* 12 */
    unsigned int unknown3;                      /* 16 */
} __attribute__ ((packed));

struct afc_connection {
    unsigned int handle;            /* 0 */
    unsigned int unknown0;          /* 4 */
    unsigned char unknown1;         /* 8 */
```

```
    unsigned char padding[3];       /* 9 */
    unsigned int unknown2;          /* 12 */
    unsigned int unknown3;          /* 16 */
    unsigned int unknown4;          /* 20 */
    unsigned int fs_block_size;     /* 24 */
    unsigned int sock_block_size;   /* 28: завжди 0x3c */
    unsigned int io_timeout;        /* 32: з AFCConnectionOpen, usu. 0 */
    void *afc_lock;                 /* 36 */
    unsigned int context;           /* 40 */
} __attribute__ ((packed));

struct afc_directory {
    unsigned char unknown[0];   /* Розмір невідомий*/
} __attribute__ ((packed));

struct afc_dictionary {
    unsigned char unknown[0];   /* Розмір невідомий*/
} __attribute__ ((packed));

typedef unsigned long long afc_file_ref;

struct usbmux_listener_1 {                  /* значення зсуву в ITunes */
    unsigned int unknown0;                  /* 0        1 */
    unsigned char *unknown1;                /* 4        ptr, можливо пристрій?
*/
    amd_device_attached_callback callback;  /* 8        _AMDDeviceAttached */
    unsigned int unknown3;                  /* 12 */
    unsigned int unknown4;                  /* 16 */
    unsigned int unknown5;                  /* 20 */
} __attribute__ ((packed));

struct usbmux_listener_2 {
    unsigned char unknown0[4144];
} __attribute__ ((packed));

struct am_bootloader_control_packet {
    unsigned char opcode;       /* 0 */
    unsigned char length;       /* 1 */
    unsigned char magic[2];     /* 2: 0x34, 0x12 */
    unsigned char payload[0];   /* 4 */
} __attribute__ ((packed));

/* ----------------------------------------------------------------------
 *   Загальні функції
 * ---------------------------------------------------------------------- */

/*
 *  Returns:
 *      MDERR_OK            У разі успіху
 *      MDERR_SYSCALL       if CFRunLoopAddSource() failed
 *      MDERR_OUT_OF_MEMORY if we ran out of memory
 */

mach_error_t AMDeviceNotificationSubscribe(am_device_notification_callback
    callback, unsigned int unused0, unsigned int unused1, void* //unsigned int
    dn_unknown3, struct am_device_notification **notification);

/*  Підключення до iPhone. Перейдіть в am_device структури, що вам дасть
повідомлення зворотного виклику .
 *
 *  Returns:
 *      MDERR_OK                якщо успішно підключений
 *      MDERR_SYSCALL           якщо setsockopt () не працює
 *      MDERR_QUERY_FAILED      якщо демон видає помилку при виконанні запиту
 *      MDERR_INVALID_ARGUMENT  якщо USBMuxConnectByPort повернув 0xffffffff
 */

mach_error_t AMDeviceConnect(struct am_device *device);
```

```
/*  Дзвінки PairingRecordPath () на цьому пристрої, перш ніж перевіряє, чи є
шлях, який існує, що повертає функція. У ході первинного підключення, шлях
повертається, що функція '/', і тому це поверне 1.
 *
 *  Повертає:
 *      0   Якщо шлях не існує
 *      1   Якщо він є
 */

CFMutableDictionaryRef AMRestoreCreateDefaultOptions(CFAllocatorRef allocator);

/* ----------------------------------------------------------------------------
 *   Недокументовані загальні функції
 * ---------------------------------------------------------------------------- */

/* режим 2 = читання, режим  3 = запис */
afc_error_t AFCFileRefOpen(struct afc_connection *conn, const char *path,
    unsigned long long mode, afc_file_ref *ref);
afc_error_t AFCFileRefSeek(struct afc_connection *conn, afc_file_ref ref,
    unsigned long long offset1, unsigned long long offset2);
afc_error_t AFCFileRefRead(struct afc_connection *conn, afc_file_ref ref,
    void *buf, unsigned int *len);
afc_error_t AFCFileRefSetFileSize(struct afc_connection *conn, afc_file_ref ref,
    unsigned long long offset);
afc_error_t AFCFileRefWrite(struct afc_connection *conn, afc_file_ref ref,
    const void *buf, unsigned int len);
afc_error_t AFCFileRefClose(struct afc_connection *conn, afc_file_ref ref);

afc_error_t AFCFileInfoOpen(struct afc_connection *conn, const char *path,
struct
    afc_dictionary **info);
afc_error_t AFCKeyValueRead(struct afc_dictionary *dict, char **key, char **
    val);
afc_error_t AFCKeyValueClose(struct afc_dictionary *dict);

unsigned int AMRestorePerformRecoveryModeRestore(struct am_recovery_device *
    rdev, CFDictionaryRef opts, void *callback, void *user_info);
unsigned int AMRestorePerformRestoreModeRestore(struct am_restore_device *
    rdev, CFDictionaryRef opts, void *callback, void *user_info);

struct am_restore_device *AMRestoreModeDeviceCreate(unsigned int unknown0,
    unsigned int connection_id, unsigned int unknown1);

unsigned int AMRestoreCreatePathsForBundle(CFStringRef restore_bundle_path,
    CFStringRef kernel_cache_type, CFStringRef boot_image_type, unsigned int
    unknown0, CFStringRef *firmware_dir_path, CFStringRef *
    kernelcache_restore_path, unsigned int unknown1, CFStringRef *
    ramdisk_path);

unsigned int AMDeviceGetConnectionID(struct am_device *device);
mach_error_t AMDeviceEnterRecovery(struct am_device *device);
mach_error_t AMDeviceDisconnect(struct am_device *device);
mach_error_t AMDeviceRetain(struct am_device *device);
mach_error_t AMDeviceRelease(struct am_device *device);
//__CFString *AMDeviceCopyValue(struct am_device *device, unsigned int, const
__CFString *cfstring);
mach_error_t AMDeviceCopyDeviceIdentifier(struct am_device *device);

mach_error_t AMDShutdownNotificationProxy(void *);

mach_error_t AMDeviceDeactivate(struct am_device *device);
mach_error_t AMDeviceActivate(struct am_device *device, CFMutableDictionaryRef);
#ifdef __cplusplus
}
#endif

#endif
```

Folder.m – робота з папками

```
      /*

     Folder.m
     iPhoneConnection

      Copyright (c) Литовченко Анна Юріївна, 2014

      */

#import "Folder.h"


@implementation Folder

-(id)initWithLocation:(NSString *)loc files:(NSArray *)fi folders:(NSArray *)fo
{

    if (self = [super init]) {
        location = [loc retain];
        [self setFiles:fi];
        folders = [fo retain];
    }
    return self;
}

-(void)dealloc {
    [location release];
    [self setFiles:nil];
    [folders release];
    [super dealloc];
}

-(NSString *)location {
    return location;
}

-(NSString *)name {
    return [location lastPathComponent];
}

-(void)setFiles:(NSArray *)fi {

     [files release];
     files = [fi retain];

}

-(NSArray *)files {
    return files;
}

-(NSArray *)folders {
    return folders;
}


@end
```

Folder.h – бібліотека для файлу Folder.m

```
      /*

      Folder.h
      iPhoneConnection

       Copyright (c) Литовченко Анна Юріївна, 2014

       */

#import <Cocoa/Cocoa.h>


@interface Folder : NSObject {

    NSArray *files;
    NSArray *folders;
    NSString *name;
    NSString *location;

}

-(id)initWithLocation:(NSString *)loc files:(NSArray *)fi folders:(NSArray *)fo;


-(NSString *)location;
-(NSString *)name;

-(void)setFiles:(NSArray *)fi;
-(NSArray *)files;
-(NSArray *)folders;


@end
```

File.m – робота з файлами

```
      /*

   File.m
   iPhoneConnection

    Copyright (c) Литовченко Анна Юріївна, 2014

    */

#import "File.h"


@implementation File

-(id)initWithLocation:(NSString *)loc {

    if (self = [super init]) {
        location = [loc retain];
    }

    return self;
}

-(void)dealloc {
    [location release];
    [super dealloc];
}

-(NSString *)location {
    return location;
}

-(NSString *)name {
    return [location lastPathComponent];
}

@end
```

File.h – бібліотека для файлу File.m

```
      /*

      File.h
      iPhoneConnection

       Copyright (c) Литовченко Анна Юріївна, 2014


       */

#import <Cocoa/Cocoa.h>


@interface File : NSObject {

    NSString *location;

}

-(id)initWithLocation:(NSString *)loc;


-(NSString *)location;
-(NSString *)name;

@end
```

```
     /*

 AFCInterface.m
 iPhoneConnection

  Copyright (c) Литовченко Анна Юріївна, 2014

  */

// AFCInterface спілкується безпосередньо з сенсорним IPod / iPhone пристроєм (
через MobileDevice.framework). AFCDevice клас переносить це за допомогою
зручного методу і функціональності.

#import "AFCInterface.h"
#import "MobileDevice.h"

@interface AFCInterface (Private)
-(NSDictionary *)createDictionaryForAFCDictionary:(struct afc_dictionary *)dict;
@end

@implementation AFCInterface


-(id)initWithAFCConnection:(struct afc_connection *)handle {

 if (self = [super init]) {
     afc_handle = handle;
 }

 return self;

}

-(BOOL)removePath:(NSString *)path {
 return AFCRemovePath(afc_handle, [path UTF8String]) == 0;
}

-(BOOL)renamePath:(NSString *)from to:(NSString *)to {
 return AFCRenamePath(afc_handle, [from UTF8String],
                              [to UTF8String]) == 0;
}

-(BOOL)createDirectory:(NSString *)path {
 return AFCDirectoryCreate(afc_handle, [path UTF8String]) == 0;
}

-(NSArray *)listFilesInPath:(NSString *)path {

 struct afc_directory *hAFCDir;

 if (AFCDirectoryOpen(afc_handle, [path UTF8String], &hAFCDir) != 0) {
     [NSException raise:@"" format:@""];
     return nil;
 } else {

     NSMutableArray *fileList = [[NSMutableArray alloc] init];
     char *buffer = nil;

     do {
           AFCDirectoryRead(afc_handle, hAFCDir, &buffer);
           if (buffer != nil) {
                 [fileList addObject:[NSString stringWithCString:buffer]];
           }
     } while (buffer != nil);
```

```
        AFCDirectoryClose(afc_handle, hAFCDir);

        return [fileList autorelease];

    }
}

-(BOOL)isDirectoryAtPath:(NSString *)path {
    NSDictionary *dict = [self getAttributesAtPath:path];
    return dict && [[dict valueForKey:@"st_ifmt"] isEqualToString:@"S_IFDIR"];
}


-(BOOL)isFileAtPath:(NSString *)path {
    NSDictionary *dict = [self getAttributesAtPath:path];
    return dict && [[dict valueForKey:@"st_ifmt"] isEqualToString:@"S_IFREG"];

}


-(NSDictionary *)getAttributesAtPath:(NSString *)path {

    struct afc_dictionary *info;

    if (AFCFileInfoOpen(afc_handle, [path UTF8String], &info) != 0) {
        return nil;
    }

    NSDictionary *fileProperties = [self createDictionaryForAFCDictionary:info];
    AFCKeyValueClose(info);


    return fileProperties;
}

-(NSDictionary *)getDeviceAttributes {

    struct afc_dictionary *info;
    if (AFCDeviceInfoOpen(afc_handle, &info) != 0) {
        return nil;
    }

    NSDictionary *deviceProperties = [self createDictionaryForAFCDictionary:info];
    AFCKeyValueClose(info);

    return deviceProperties;
}


-(unsigned long long)openFileAtPath:(NSString *)path withMode:(int)mode {

    afc_file_ref rAFC;

    int ret = AFCFileRefOpen(afc_handle, [path UTF8String], mode, &rAFC);
    if (ret != 0) {
        NSLog(@"AFCFileRefOpen(%d): %d", mode, ret);
        return 0;
    }
    return rAFC;
}


-(BOOL)closeFile:(unsigned long long)rAFC {
    return AFCFileRefClose(afc_handle, rAFC) == 0;
}
```

```
-(NSData *)readFromFile:(unsigned long long)rAFC size:(unsigned int *)size
offset:(off_t)offset {

  int ret = AFCFileRefSeek(afc_handle, rAFC, offset, 0);

  if (ret != 0) {
      return nil;
  }

    char *buffer = malloc(*size);
    unsigned int s = *size;

  ret = AFCFileRefRead(afc_handle, rAFC, buffer, &s);

  if (ret != 0) {
        //buffer = nil;
        NSLog(@"ret: %d", ret);
      return nil;
  }

    *size = s;

    NSData *data = [NSData dataWithBytes:buffer length:s];

    free(buffer);

    //buffer = buf;

  return data;
}


-(BOOL)writeToFile:(unsigned long long)rAFC data:(const char *)data
size:(size_t)size offset:(off_t)offset {

  if (size > 0) {

      int ret = AFCFileRefSeek(afc_handle, rAFC, offset, 0);

      if (ret != 0) {
            return NO;
      }

      ret = AFCFileRefWrite(afc_handle, rAFC, data, (unsigned long)size);

      if (ret != 0) {
            return NO;
      }
  }

  // Записувати 0 байт неможливо.
  return YES;
}

-(BOOL)setSizeOfFile:(NSString *)path toSize:(off_t)size {

  afc_file_ref rAFC;
  int ret = AFCFileRefOpen(afc_handle, [path UTF8String], 3, &rAFC);

  if (ret != 0) {
      return NO;
  }

  ret = AFCFileRefSetFileSize(afc_handle, rAFC, size);
  AFCFileRefClose(afc_handle, rAFC);

  if (ret != 0) {
      return NO;
  }
```

```
  return YES;

}

-(NSDictionary *)createDictionaryForAFCDictionary:(struct afc_dictionary *)dict
{

  NSMutableDictionary *dictionary = [[NSMutableDictionary alloc] init];
  char *key, *val;

  while ((AFCKeyValueRead(dict, &key, &val) == 0) && key && val) {
       [dictionary setValue:[NSString stringWithCString:val] forKey:[NSString
stringWithCString:key]];
  }

  return [dictionary autorelease];

}


@end
```

AFCInterface.h – бібліотека для файлу AFCInterface.m

```
      /*

     AFCInterface.h
     iPhoneConnection

      Copyright (c) Литовченко Анна Юріївна, 2014

      */

#import <Cocoa/Cocoa.h>
#import "MobileDevice.h"

@interface AFCInterface : NSObject {

     struct afc_connection *afc_handle;

}

-(id)initWithAFCConnection:(struct afc_connection *)handle;

-(BOOL)removePath:(NSString *)path;
-(BOOL)renamePath:(NSString *)from to:(NSString *)to;
-(BOOL)createDirectory:(NSString *)path;
-(NSArray *)listFilesInPath:(NSString *)path;
-(BOOL)isDirectoryAtPath:(NSString *)path;
-(BOOL)isFileAtPath:(NSString *)path;
-(NSDictionary *)getAttributesAtPath:(NSString *)path;
-(NSDictionary *)getDeviceAttributes;
-(unsigned long long)openFileAtPath:(NSString *)path withMode:(int)mode;
-(BOOL)closeFile:(unsigned long long)rAFC;
//-(BOOL)readFromFile:(unsigned long long)rAFC buffer:(char *)buffer
size:(unsigned int *)size offset:(off_t)offset;
-(NSData *)readFromFile:(unsigned long long)rAFC size:(unsigned int *)size
offset:(off_t)offset;

-(BOOL)writeToFile:(unsigned long long)rAFC data:(const char *)data
size:(size_t)size offset:(off_t)offset;
-(BOOL)setSizeOfFile:(NSString *)path toSize:(off_t)size ;
@end
```

MainWindowController.m - основна програма

```
      /*

      MainWindowController.m
      Підключення iPhone

       Copyright (c) Литовченко Анна Юріївна, 2014
       */


#import "MainWindowController.h"
#import "AFCFactory.h"
#import "File.h"
#import "Folder.h"
#import "ImageAndTextCell.h"

@implementation MainWindowController



#pragma mark -
#pragma mark Normal Obj-C Stuff

-(void)awakeFromNib {

      [[AFCFactory factory] setDelegate:self];

      [disconnectWarning setFrame:NSMakeRect(0, 0, [[self window]
frame].size.width, [[self window] frame].size.height)];
      [[[self window] contentView] addSubview:disconnectWarning];
      [disconnectWarning setHidden:YES];

      [contentBox setContentView:noSelectionView];

      [self setupToolbar];


}

-(void)dealloc {
      [root release];
      [iPhone release];
      [super dealloc];
}

#pragma mark Interface actions

-(IBAction)getFiles:(id)sender {

      [NSApp beginSheet:loadingSheet modalForWindow:[self window]
modalDelegate:nil
         didEndSelector:nil contextInfo:nil];

      [root release];
    root = [[self folderAtPath:@"/"] retain];

    [fileList reloadItem:nil];

      [NSApp endSheet:loadingSheet];
      [loadingSheet orderOut:nil];

}


-(IBAction)showDeviceInfo:(id)sender {
```

```
      NSRunAlertPanelRelativeToWindow(@"Інформація про пристрій", [[[iPhone
deviceInterface] getDeviceAttributes] description], @"OK", nil, nil, [self
window]);


}

-(IBAction)deleteFile:(id)sender {

      id item;
      item = [fileList itemAtRow:[fileList selectedRow]];
      if ([item isKindOfClass:[File class]]) {

            int ret = NSRunAlertPanelRelativeToWindow([NSString
stringWithFormat:@"Ви впевнені, що хочете видалити файл %@?",

[[(File *) item location] lastPathComponent]],
                                                                  @"Це не
може бути скасовано, і видалення неправильно файлу може порушити
функціональність пристрою.",

@"Delete",

@"Cancel",
                                                                  nil,
                                                                  [self
window]);

            if (ret == NSAlertDefaultReturn) {

                  [iPhone deleteFileAtPath:[(File *)item location]];

                  Folder *parent = [fileList parentForItem:item];

                  if (!parent) {
                        parent = root;
                  }

                  [parent setFiles:[self filesAtPath:[parent location]]];
                  [fileList reloadData];
                  [self updateSpace];
            }

      }
}

-(IBAction)addFile:(id)sender {

      NSOpenPanel *openPanel = [NSOpenPanel openPanel];

      [openPanel setCanChooseFiles:YES];
      [openPanel setCanChooseDirectories:NO];

      [openPanel beginSheetForDirectory:nil file:nil modalForWindow:[self
window] modalDelegate:self

didEndSelector:@selector(choosePath:returnCode:contextInfo:) contextInfo:nil];

}



-(void)choosePath:(NSOpenPanel *)sheet returnCode:(int)returnCode
contextInfo:(void*)contextInfo {
      if (returnCode == NSOKButton) {

            NSString *path = @"/";
```

```
            id item;
            item = [fileList itemAtRow:[fileList selectedRow]];
            if ([item isKindOfClass:[Folder class]]) {
                  path = [(Folder *)item location];
            } else {
                  item = root;
            }

            [iPhone createFileAtPath:[path
stringByAppendingPathComponent:[[sheet filename] lastPathComponent]]

             withData:[NSData dataWithContentsOfFile:[sheet filename]]];


            [(Folder *)item setFiles:[self filesAtPath:path]];
            [fileList reloadData];
            [self updateSpace];

      }
}

-(IBAction)extractFile:(id)sender {

      id item;
      item = [fileList itemAtRow:[fileList selectedRow]];
      if ([item isKindOfClass:[File class]]) {

            NSSavePanel *savePanel = [NSSavePanel savePanel];

            [savePanel setCanCreateDirectories:YES];

            File *file = item;

            [savePanel beginSheetForDirectory:nil file:[[file location]
lastPathComponent]  modalForWindow:[self window]
                                          modalDelegate:self
didEndSelector:@selector(savePath:returnCode:contextInfo:) contextInfo:file];

      }
}

-(void)savePath:(NSSavePanel *)sheet returnCode:(int)returnCode
contextInfo:(void*)contextInfo {
      if (returnCode == NSOKButton && iPhone) {

            if ([(id)contextInfo isKindOfClass:[File class]]) {

                  [[iPhone contentsOfFileAtPath:[(File *)contextInfo location]]
writeToFile:[sheet filename] atomically:YES];

            }
      }
}


#pragma mark Worker methods

-(Folder *)folderAtPath:(NSString *)path {

    NSMutableArray *folders = [[NSMutableArray alloc] init];


    NSArray *origFolders = [iPhone listOfFoldersAtPath:path];
    NSEnumerator *e = [origFolders objectEnumerator];
    NSString *fileName;

    while (fileName = [e nextObject]) {
        [folders addObject:[self folderAtPath:[path
stringByAppendingPathComponent:fileName]]];
```

```
    }

    return [[[Folder alloc] initWithLocation:path files:[self filesAtPath:path]
folders:[folders autorelease]] autorelease];

}

-(NSArray *)filesAtPath:(NSString *)path {

      NSMutableArray *files = [[NSMutableArray alloc] init];
      NSArray *origFiles = [iPhone listOfFilesAtPath:path];

    NSEnumerator *e = [origFiles objectEnumerator];
    NSString *fileName;

    while (fileName = [e nextObject]) {
        [files addObject: [[[File alloc] initWithLocation:[path
stringByAppendingPathComponent:fileName]] autorelease]];
    }

      return [files autorelease];

}

-(void)updateSpace {

      if (iPhone) {
      NSDictionary *props = [[iPhone deviceInterface] getDeviceAttributes];

      unsigned long totalSpace = [[props valueForKey:@"FSTotalBytes"]
floatValue] / 1024;
      unsigned long freeSpace = [[props valueForKey:@"FSFreeBytes"] floatValue]
/ 1024;
      unsigned long usedSpace = totalSpace - freeSpace;

      float percent = ((float)usedSpace / (float)totalSpace) * 10.0f;

      [(NSLevelIndicator *)usedSpaceView setFloatValue:percent];

      [usedSpaceItem setLabel:[NSString stringWithFormat:@"%1.2fGb of %1.2fGb
free", ((float)freeSpace)/1024/1024, ((float)totalSpace)/1024/1024]];

      } else {
            [(NSLevelIndicator *)usedSpaceView setFloatValue:0.0];
            [usedSpaceItem setLabel:@"Дисковий простір"];
      }
}

-(BOOL)extensionIsImage:(NSString *)ext {

      return ( [ext caseInsensitiveCompare:@"jpg"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"gif"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"png"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"tif"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"tiff"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"thm"] == NSOrderedSame);

}

-(BOOL)extensionIsQTMedia:(NSString *)ext {

      return ( [ext caseInsensitiveCompare:@"mov"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"mp3"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"wav"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"aif"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"aiff"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"aifc"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"mp2"] == NSOrderedSame ||
                  [ext caseInsensitiveCompare:@"mpg"] == NSOrderedSame ||
```

```
                [ext caseInsensitiveCompare:@"mpga"] == NSOrderedSame ||
                [ext caseInsensitiveCompare:@"aa"] == NSOrderedSame ||
                [ext caseInsensitiveCompare:@"mov"] == NSOrderedSame ||
                [ext caseInsensitiveCompare:@"m4a"] == NSOrderedSame ||
                [ext caseInsensitiveCompare:@"m4p"] == NSOrderedSame ||
                [ext caseInsensitiveCompare:@"m4b"] == NSOrderedSame ||
                [ext caseInsensitiveCompare:@"m4v"] == NSOrderedSame ||
                [ext caseInsensitiveCompare:@"m4r"] == NSOrderedSame);

}

#pragma mark -
#pragma mark AFC Factory Delegates


-(void)AFCDeviceWasConnected:(AFCDeviceRef *)dev {

      // Це те, де відбувається дія - виявлено новий iPhone / IPod Touch.

      if (iPhone) {

            if ([[[iPhone device] serialNumber] isEqualToString:[dev
serialNumber]]) {

                  // Коли ж пристрій підключений, це для різних device_id. Таким
чином, інформувати, що пристрої були змінені.

                  [iPhone setDeviceRef:dev];

                  [[disconnectWarning animator] setAlphaValue:0.0];
                  [disconnectWarning setHidden:YES];

                  return;

            }

      }

      // Якщо це не той же саме пристрій, як і попередній, робить новий
пристрій.

      [iPhone release];
      iPhone = nil;

      iPhone = [[AFCDevice alloc] initWithRef:dev];

      if (!iPhone) {

            [NSException raise:@"MainWindowController" format:@"Сталася помилка
при спробі ініціалізації пристрою."];

      } else {

            [iPhone setDelegate:self];
            [self updateSpace];
            [self getFiles:nil];

            [[disconnectWarning animator] setAlphaValue:0.0];
            [disconnectWarning setHidden:YES];
      }


}


#pragma mark -
#pragma mark AFCDevice Delegates

-(void)deviceWasDisconnected:(AFCDevice *)device {
```

```
      [disconnectWarning setHidden:NO];
      [[disconnectWarning animator] setAlphaValue:1.0];

}



#pragma mark -
#pragma mark OutlineView Delegates

- (BOOL)outlineView:(NSOutlineView *)outlineView shouldSelectItem:(id)item {

      return YES;
}

- (int)outlineView:(NSOutlineView *)outlineView numberOfChildrenOfItem:(id)item
{
    if (item) {
            if ([item isKindOfClass: [Folder class]]) {
                  return ([[item files] count] + [[item folders] count]);
            } else if ([item isKindOfClass:[File class]]) {
                  return 0;
            }
      }

      // якщо значення дорівнює нулю, то це кореневий рівень

    return [[root files] count] + [[root folders] count];

}

- (BOOL)outlineView:(NSOutlineView *)outlineView isItemExpandable:(id)item
{
    if (item) {
            if ([item isKindOfClass: [Folder class]]) {
                  return ([[item files] count] + [[item folders] count]) > 0;
            } else {
                  return NO;
            }
      }
      return NO;
}

- (id)outlineView:(NSOutlineView *)outlineView child:(int)index ofItem:(id)item
{
    if (item) {
            if ([item isKindOfClass: [Folder class]]) {

            // Файли у папці

            if (index >= [[item files] count]) {
                return [[item folders] objectAtIndex: index - [[item files]
count]];
            } else {
                return [[item files] objectAtIndex:index];
            }

            }
      } else {
            //root

        if (index >= [[root files] count]) {
            return [[root folders] objectAtIndex: index - [[root files] count]];
        } else {
            return [[root files] objectAtIndex:index];
        }
```

```
      }

    return nil;
}

- (void)outlineViewSelectionIsChanging:(NSNotification *)notification {


      id item;
      item = [[notification object] itemAtRow:[[notification object]
selectedRow]];
      if ([item isKindOfClass:[File class]]) {

            if ([self extensionIsImage:[[(File *)item location] pathExtension]])
{

             [imageView setImage:[[[NSImage alloc] initWithData:[iPhone
contentsOfFileAtPath:[(File *)item location]]] autorelease]];
                  [contentBox setContentView:imageView];


            } else if ([self extensionIsQTMedia:[[(File *)item location]
pathExtension]]) {

                  [NSApp beginSheet:loadingSheet modalForWindow:[self window]
modalDelegate:nil
                     didEndSelector:nil contextInfo:nil];

                  NSData *data = [iPhone contentsOfFileAtPath:[(File *)item
location]];

                  QTDataReference *ref = [QTDataReference
dataReferenceWithReferenceToData:data

                                          name:[[(File *)item location]
lastPathComponent]

                                      MIMEType:nil];
                  NSError *err = nil;
                  QTMovie *mov = [[QTMovie alloc] initWithDataReference:ref
error:&err];

                  [NSApp endSheet:loadingSheet];
                  [loadingSheet orderOut:nil];


                  if (mov) {

                        [qtView setMovie:[mov autorelease]];
                        [contentBox setContentView:qtView];

                  } else {

                       NSRunAlertPanelRelativeToWindow([NSString
stringWithFormat:@"Не вдалося завантажити %@.", [[(File *)item location]
lastPathComponent]],
                                                                        [err
description],

                                                                        @"OK",
                                                                        nil,
                                                                        nil,
                                                                        [self
window]);

                  }
```

```
            } else {
                  [textView setString:[[[NSString alloc] initWithData:[iPhone
contentsOfFileAtPath:[(File *)item location]]

encoding:[NSString defaultCStringEncoding]] autorelease]];

                  [contentBox setContentView:textContainer];

            }

      } else {
            [contentBox setContentView:noSelectionView];
    }

}

- (void)outlineView:(NSOutlineView *)olv willDisplayCell:(NSCell *)cell
forTableColumn:(NSTableColumn *)tableColumn item:(id)item {
      // Викликається якраз перед  звертанням до осередку

      if ([item isKindOfClass:[Folder class]]) {
            NSImage *im = [[NSWorkspace sharedWorkspace]
iconForFile:@"/Volumes/"];
            [im setSize:NSMakeSize(16, 16)];
            [(ImageAndTextCell *)cell setImage:im];
      } else {
            NSImage *im = [[NSWorkspace sharedWorkspace] iconForFileType:[[(File
*)item location] pathExtension]];
            [im setSize:NSMakeSize(16, 16)];

            [(ImageAndTextCell *)cell setImage:im];
      }

}


-(id)outlineView:(NSOutlineView *)outlineView
objectValueForTableColumn:(NSTableColumn *)tableColumn byItem:(id)item {

      if ([item isKindOfClass:[Folder class]]) {
            return [item name];
      } else if ([item isKindOfClass:[File class]]) {
        return [item name];
    }

      return @"Unknown item";

}

- (BOOL)outlineView:(NSOutlineView *)outlineView writeItems:(NSArray *)items
toPasteboard:(NSPasteboard *)pboard {

      // Для drag/drop
      return NO;
}


#pragma mark -
#pragma mark Toolbar Setup


- (NSToolbarItem *)toolbar:(NSToolbar *)toolbar
       itemForItemIdentifier:(NSString *)itemIdentifier
 willBeInsertedIntoToolbar:(BOOL)flag
{
      NSToolbarItem *item = [[NSToolbarItem alloc]
initWithItemIdentifier:itemIdentifier];
```

```
      if ( [itemIdentifier isEqualToString:@"InfoItem"] ) {

            [item setLabel:@"Інформація про пристрій"];
            [item setPaletteLabel:[item label]];
            [item setTarget:self];
            [item setImage:[NSImage imageNamed:@"Info"]];
            [item setAction:@selector(showDeviceInfo:)];

      } else if ( [itemIdentifier isEqualToString:@"DeleteItem"] ) {

            [item setLabel:@"Видалення файлу"];
            [item setPaletteLabel:[item label]];
            [item setTarget:self];
            [item setImage:[NSImage imageNamed:@"Delete"]];
            [item setAction:@selector(deleteFile:)];

      } else if ( [itemIdentifier isEqualToString:@"AddFileItem"] ) {

            [item setLabel:@"Додати файл..."];
            [item setPaletteLabel:[item label]];
            [item setTarget:self];
            [item setImage:[NSImage imageNamed:@"Add"]];
            [item setAction:@selector(addFile:)];

      } else if ( [itemIdentifier isEqualToString:@"ExtractFileItem"] ) {

            [item setLabel:@"Розпакувати файл..."];
            [item setPaletteLabel:[item label]];
            [item setTarget:self];
            [item setImage:[NSImage imageNamed:@"Copy"]];
            [item setAction:@selector(extractFile:)];

      } else if ([itemIdentifier isEqualToString:@"UsedSpaceItem"] ) {

            [item release];
            return usedSpaceItem;


      }

      return [item autorelease];
}

- (BOOL)validateToolbarItem:(NSToolbarItem *)theItem
{
      if ([[theItem itemIdentifier] isEqualToString:@"InfoItem"]) {
            return (iPhone != nil);
      } else if ([[theItem itemIdentifier] isEqualToString:@"DeleteItem"]) {
            return [[fileList itemAtRow:[fileList selectedRow]]
isKindOfClass:[File class]];
      } else if ([[theItem itemIdentifier] isEqualToString:@"AddFileItem"]) {
            return (iPhone != nil);
      } else if ([[theItem itemIdentifier] isEqualToString:@"ExtractFileItem"])
{
            return [[fileList itemAtRow:[fileList selectedRow]]
isKindOfClass:[File class]];
      } else {
            return YES;
      }
}

- (NSArray *)toolbarAllowedItemIdentifiers:(NSToolbar*)toolbar
{
      return [NSArray arrayWithObjects:NSToolbarSeparatorItemIdentifier,
                  NSToolbarSpaceItemIdentifier,
                  NSToolbarFlexibleSpaceItemIdentifier,
                  NSToolbarCustomizeToolbarItemIdentifier,
                  @"InfoItem",
                  @"DeleteItem",
```

```
                    @"AddFileItem",
                    @"ExtractFileItem",
                    @"UsedSpaceItem",
                    nil];
}

- (NSArray *)toolbarDefaultItemIdentifiers:(NSToolbar*)toolbar
{
      return [NSArray arrayWithObjects:@"DeleteItem", @"ExtractFileItem",
@"AddFileItem", NSToolbarFlexibleSpaceItemIdentifier,
                    @"UsedSpaceItem", NSToolbarFlexibleSpaceItemIdentifier,
@"InfoItem", nil];
}

-(void) setupToolbar {


      usedSpaceItem = [[NSToolbarItem alloc]
initWithItemIdentifier:@"UsedSpaceItem"];
      [usedSpaceItem setView:usedSpaceView];
      [usedSpaceItem setLabel:@"Дисковий простір"];
      [usedSpaceItem setPaletteLabel:@"Дисковий простір"];

      NSToolbar *toolbar = [[NSToolbar alloc]
initWithIdentifier:@"iPhoneBrowser.Toolbar"];
    //[toolbar autorelease];
    [toolbar setDelegate:self];
    [toolbar setAllowsUserCustomization:YES];
    [toolbar setAutosavesConfiguration:YES];
    [[self window] setToolbar:[toolbar autorelease]];

}


@end
```

MainWindowController.h – бібліотека для основної програми

```
      /*

      MainWindowController.h
      iPhoneConnection

       Copyright (c) Литовченко Анна Юріївна, 2014
       */

#import <Cocoa/Cocoa.h>
#import "MobileDevice.h"
#import "AFCDevice.h"
#import "Folder.h"
#import <QTKit/QTKit.h>

@interface MainWindowController : NSWindowController {

      IBOutlet NSOutlineView *fileList;
      IBOutlet NSTextField *pathField;
    IBOutlet NSImageView *imageView;
      IBOutlet NSTextView *textView;
      IBOutlet NSScrollView *textContainer;
      IBOutlet NSBox *contentBox;
      IBOutlet NSView *noSelectionView;
      IBOutlet QTMovieView *qtView;
      IBOutlet NSPanel *loadingSheet;

      IBOutlet NSView *usedSpaceView;
      IBOutlet NSView *disconnectWarning;

      NSToolbarItem *usedSpaceItem;

      Folder *root;
      AFCDevice *iPhone;
}


-(IBAction)getFiles:(id)sender;
-(IBAction)deleteFile:(id)sender;
-(IBAction)addFile:(id)sender;
-(IBAction)extractFile:(id)sender;
-(IBAction)showDeviceInfo:(id)sender;

-(Folder *)folderAtPath:(NSString *)path;
-(NSArray *)filesAtPath:(NSString *)path;

-(void)setupToolbar;
-(void)updateSpace;
-(BOOL)extensionIsImage:(NSString *)ext;
-(BOOL)extensionIsQTMedia:(NSString *)ext;

@end
```

ImageAndTextCell.m – робота з текстом та зображеннями

```
   /*
    ImageAndTextCell.m
    Copyright (c) Литовченко Анна Юріївна, 2014
*/

#import "ImageAndTextCell.h"
#import <AppKit/NSCell.h>

@implementation ImageAndTextCell

- (id)init {
    if (self = [super init]) {
        [self setLineBreakMode:NSLineBreakByTruncatingTail];
        [self setSelectable:YES];
    }
    return self;
}

- (void)dealloc {
    [image release];
    [super dealloc];
}

- (id)copyWithZone:(NSZone *)zone {
    ImageAndTextCell *cell = (ImageAndTextCell *)[super copyWithZone:zone];
    // Зображення буде скопійовано, і ми повинні зберегти або скопіювати його.
    cell->image = [image retain];
    return cell;
}

- (void)setImage:(NSImage *)anImage {
    if (anImage != image) {
        [image release];
        image = [anImage retain];
    }
}

- (NSImage *)image {
    return image;
}

- (NSRect)imageRectForBounds:(NSRect)cellFrame {
    NSRect result;
    if (image != nil) {
        result.size = [image size];
        result.origin = cellFrame.origin;
        result.origin.x += 3;
        result.origin.y += ceil((cellFrame.size.height - result.size.height) /
2);
    } else {
        result = NSZeroRect;
    }
    return result;
}

- (NSRect)titleRectForBounds:(NSRect)cellFrame {
    NSRect result;
    if (image != nil) {
        CGFloat imageWidth = [image size].width;
        result = cellFrame;
        result.origin.x += (3 + imageWidth);
        result.size.width -= (3 + imageWidth);
    } else {
        result = NSZeroRect;
    }
```

```
    return result;
}


- (void)editWithFrame:(NSRect)aRect inView:(NSView *)controlView editor:(NSText
*)textObj delegate:(id)anObject event:(NSEvent *)theEvent {
    NSRect textFrame, imageFrame;
    NSDivideRect (aRect, &imageFrame, &textFrame, 3 + [image size].width,
NSMinXEdge);
    [super editWithFrame: textFrame inView: controlView editor:textObj
delegate:anObject event: theEvent];
}

- (void)selectWithFrame:(NSRect)aRect inView:(NSView *)controlView
editor:(NSText *)textObj delegate:(id)anObject start:(NSInteger)selStart
length:(NSInteger)selLength {
    NSRect textFrame, imageFrame;
    NSDivideRect (aRect, &imageFrame, &textFrame, 3 + [image size].width,
NSMinXEdge);
    [super selectWithFrame: textFrame inView: controlView editor:textObj
delegate:anObject start:selStart length:selLength];
}

- (void)drawWithFrame:(NSRect)cellFrame inView:(NSView *)controlView {
    if (image != nil) {
        NSRect    imageFrame;
        NSSize imageSize = [image size];
        NSDivideRect(cellFrame, &imageFrame, &cellFrame, 3 + imageSize.width,
NSMinXEdge);
        if ([self drawsBackground]) {
            [[self backgroundColor] set];
            NSRectFill(imageFrame);
        }
        imageFrame.origin.x += 3;
        imageFrame.size = imageSize;

        if ([controlView isFlipped])
            imageFrame.origin.y += ceil((cellFrame.size.height +
imageFrame.size.height) / 2);
        else
            imageFrame.origin.y += ceil((cellFrame.size.height -
imageFrame.size.height) / 2);

        [image compositeToPoint:imageFrame.origin
operation:NSCompositeSourceOver];
    }
    [super drawWithFrame:cellFrame inView:controlView];
}

- (NSSize)cellSize {
    NSSize cellSize = [super cellSize];
    cellSize.width += (image ? [image size].width : 0) + 3;
    return cellSize;
}

- (NSUInteger)hitTestForEvent:(NSEvent *)event inRect:(NSRect)cellFrame
ofView:(NSView *)controlView {
    NSPoint point = [controlView convertPoint:[event locationInWindow]
fromView:nil];
    // Якщо у нас є зображення, ми повинні бачити, якщо користувач натиснув на
частину зображення.
    if (image != nil) {
        // Цей код точно імітує drawWithFrame:inView:
        NSSize imageSize = [image size];
        NSRect imageFrame;
        NSDivideRect(cellFrame, &imageFrame, &cellFrame, 3 + imageSize.width,
NSMinXEdge);

        imageFrame.origin.x += 3;
```

```
        imageFrame.size = imageSize;
        //
        if (NSMouseInRect(point, imageFrame, [controlView isFlipped])) {

            return NSCellHitContentArea;
        }
    }
        return [super hitTestForEvent:event inRect:cellFrame
ofView:controlView];
}


@end
```

ImageAndTextCell.h – бібліотека для файлу ImageAndTextCell.m

```
#import <Cocoa/Cocoa.h>

@interface ImageAndTextCell : NSTextFieldCell {
@private
    NSImage *image;
}

- (void)setImage:(NSImage *)anImage;
- (NSImage *)image;

- (void)drawWithFrame:(NSRect)cellFrame inView:(NSView *)controlView;
- (NSSize)cellSize;

@end
```

DisconnectedView.m – Розєднання

```
      /*

      DisconnectedView.m
      iPhoneConnection

       Copyright (c) Литовченко Анна Юріївна, 2014

       */

#import "DisconnectedView.h"


@implementation DisconnectedView

- (id)initWithFrame:(NSRect)frame {
    self = [super initWithFrame:frame];
    if (self) {
        // Код ініціалізації тут.
    }
    return self;
}

- (void)drawRect:(NSRect)rect {

      NSRect bounds = [self bounds];


      [[[NSColor blackColor] colorWithAlphaComponent:0.4] set];

      NSSize boxSize = NSMakeSize(bounds.size.width, bounds.size.height);

      NSRect boxRect = NSMakeRect((bounds.size.width / 2) - (boxSize.width / 2),
                                  (bounds.size.height / 2) -
(boxSize.height / 2),
                                  boxSize.width,
                                  boxSize.height);

      [NSBezierPath fillRect:boxRect];

      NSImage *im = [NSImage imageNamed:@"Disconnected"];

      [im drawAtPoint:NSMakePoint((bounds.size.width / 2) - ([im size].width /
2),
                                  (bounds.size.height / 2) - ([im
size].height / 2))
             fromRect:NSZeroRect
            operation:NSCompositeSourceOver
             fraction:1.0];

}

- (void)mouseDown:(NSEvent *)theEvent {


}

@end
```

DisconnectedView.h – бібліотека для файлу DisconnectedView.m

```
/*

 DisconnectedView.h
 iPhoneConnection

 Copyright (c) Литовченко Анна Юріївна, 2014

 */

#import <Cocoa/Cocoa.h>


@interface DisconnectedView : NSView {

}

@end
```

AFCFactory.m – підключення / відключення повідомлень

```
	/*

	AFCFactory.m
	iPhoneConnection

	 Copyright (c) Литовченко Анна Юріївна, 2014
	 All rights reserved.


	 */

// AFCFActory складається з одного класу, який визначає підключення /
відключення повідомлень від MobileDevice.framework, і повідомляє його
представника, коли вони відбуваються. Встановити делегування наступним чином:
//
// [[AFCFactory factory] setDelegate:self];
//
// -(void)AFCDeviceWasConnected:(AFCDeviceRef *)dev;

#import "AFCFactory.h"
#import "MobileDevice.h"
#import "AFCDeviceRef.h"

@interface AFCFactory (Private)

-(void)notify:(struct am_device_notification_callback_info *)info;

@end

@implementation AFCFactory


#pragma mark -
#pragma mark C -> Obj-C callback delegates

static void notify_callback(struct am_device_notification_callback_info *info,
void* arg) {
	AFCFactory* fact = (AFCFactory *)arg;
	[fact notify:info];

}

#pragma mark -
#pragma mark Obj-C

static AFCFactory *factory;

+(AFCFactory *)factory {

	if (!factory) {
		factory = [[AFCFactory alloc] init];
	}

	return factory;
}

-(id)init {

	if (factory) {
		[self release];
		return factory;
	} else {

		if (self = [super init]) {
```

```
                struct am_device_notification *notif;
                int ret = AMDeviceNotificationSubscribe(notify_callback, 0, 0,
self,

      &notif);
                if (ret != 0) {
                      [NSException raise:@"AFCFactory"
format:@"AMDeviceNotificationSubscribe з помилкою %d", ret];

                      [self release];
                      return nil;
                }

           }

           return self;
      }
}

-(void)dealloc {
      [self setDelegate:nil];
      [super dealloc];
}

-(void)setDelegate:(id)del {

      [delegate release];
      delegate = [del retain];
}

-(id)delegate {
      return delegate;
}


// Викликається статичний допоміжний метод notify_callback
-(void)notify:(struct am_device_notification_callback_info *)info {

      if (info->msg == ADNCI_MSG_CONNECTED) {

           if ([delegate respondsToSelector:@selector(AFCDeviceWasConnected:)])
{
                [delegate performSelector:@selector(AFCDeviceWasConnected:)
                                     withObject: [[[AFCDeviceRef alloc]
initWithAFCDeviceStruct:info->dev] autorelease]];
                }

    } else if(info->msg == ADNCI_MSG_DISCONNECTED) {

           [[NSNotificationCenter defaultCenter]
postNotificationName:@"AFC_DeviceWasDisconnected"

           object:[NSNumber numberWithUnsignedInt:info->dev->device_id]];

           /*if ([delegate
respondsToSelector:@selector(AFCDeviceWasDisconnected:)]) {
                [delegate performSelector:@selector(AFCDeviceWasDisconnected:)
                                     withObject: [[[AFCDeviceRef alloc]
initWithAFCDeviceStruct:info->dev] autorelease]];
           }*/



    }

}
@end
```

AFCFactory.h – бібліотека для файлу AFCFactory.m

```
	/*

	AFCFactory.h
	iPhoneConnection


	 */

#import <Cocoa/Cocoa.h>


@interface AFCFactory : NSObject {

	id delegate;

}


+(AFCFactory *)factory;


-(void)setDelegate:(id)del;
-(id)delegate;


@end
```

AFCDeviceRef.m – клас-оболонка для структури am_device

```
      /*

      AFCDeviceRef.m
      iPhoneConnection

       Copyright (c) Литовченко Анна Юріївна, 2014
       */

// AFCDeviceRef є досить простий клас-оболонка для структури am_device, і дозволяє структури, які передаються через делегати. Він також включає пару допоміжних методів для отримання даних.

#import "AFCDeviceRef.h"

@implementation AFCDeviceRef

-(id)initWithAFCDeviceStruct:(struct am_device *)dev {

      if (self = [super init]) {
       device = dev;
      }

      return self;
}


-(NSString *)serialNumber {
      return [NSString stringWithCString:device->serial];
}

-(unsigned int)productID {
      return device->product_id;
}

-(unsigned int)deviceID {
      return device->device_id;
}

-(struct am_device *)device {
      return device;
}

@end
```

AFCDeviceRef.h – бібліотека для файлу AFCDeviceRef.m

```
	/*

	AFCDeviceRef.h
	iPhoneConnection

	 Copyright (c) Литовченко Анна Юріївна, 2014

	 */

#import <Cocoa/Cocoa.h>
#import "MobileDevice.h"

@interface AFCDeviceRef : NSObject {

	struct am_device *device;

}

-(id)initWithAFCDeviceStruct:(struct am_device *)dev;

-(NSString *)serialNumber;
-(unsigned int)productID;
-(unsigned int)deviceID;

-(struct am_device *)device;


@end
```