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 управління файлами на iPhone”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-22М-1
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Коваленко В.О.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Доренський О.П.
«____» _____ 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

Коваленко Владиславу Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи управління файлами на iPhone*

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

затверджені наказом вищого навчального закладу № 32-13 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 управління файлами на iPhone*

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

Коваленко В.О. Дослідження та програмна реалізація системи управління файлами на iPhone. 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 управління файлами на iPhone.

Метою розробки є дослідження та програмна реалізація системи управління файлами на iPhone.

Об'єктом дослідження є процес управління файлами на iPhone.

Предметом дослідження є методи управління файлами на iPhone.

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

Результат роботи – програмна реалізація системи управління файлами на iPhone.

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 мобільному пристрої під керуванням iOS..

Програму розроблено в середовищі Objective C.

Ключові слова: комп'ютерні науки, управління файлами, iPhone

ABSTRACT

Kovalenko V.O. Research and software implementation of the iPhone file management 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 file management system on the iPhone.

The purpose of the development is the research and software implementation of the file management system on the iPhone.

The object of the study is the process of managing files on the iPhone.

The subject of the study is methods of managing files on the iPhone.

Research methods are based on the methods of the theory of construction of operating systems, methods of mathematical statistics, methods of software development.

The result of the work is a software implementation of the file management system on the iPhone.

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

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

The program can be used on a mobile device running iOS.

The program was developed in the Objective C environment.

Keywords: computer science, file management, iPhone

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. Система призначена для управління файлами на iPhone, який працює під керуванням ОС Apple iOS. Для цього використовується файловий менеджер. Файловий менеджер – комп'ютерна програма, що надає інтерфейс користувача для роботи з файловою системою й файлами. Файловий менеджер дозволяє виконувати найбільш часті операції над файлами – створення, відкриття/програвання/перегляд, редагування, переміщення, перейменування, копіювання, видалення, зміна атрибутів і властивостей, пошук файлів і призначення прав. Крім основних функцій, багато файлових менеджерів включають ряд додаткових можливостей, наприклад, таких як робота з мережею (через FTP, NFS і т.п.), резервне копіювання, керування принтерами та ін.

Виділяють різні типи файлових менеджерів: навігаційн і просторові – іноді підтримується перемикання між цими режимами. Інший вид – двопанельні – у загальному випадку мають дві рівноцінних панелі для списку файлів, дерева каталогів і т.п. На даний момент дуже широке розповсюдження отримали пристрої, які працюють під операційною системою Apple iOS. Apple iOS – мобільна операційна система, розроблена компанією Apple на основі Mac OS X для смартфонів iPhone, медіаплеєрів iPod Touch і планшетних комп'ютерів iPad. Була випущена 29 червня 2007 року, з тих пор регулярно обновляється. Входить у сімейство операційних систем Apple OS X, до якого також відноситься й операційна для настільних комп'ютерів – Mac OS X. Операційна система Apple iOS також працює на Apple TV, але з користувальницьким інтерфейсом, що відрізняється.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи управління файлами на iPhone.

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

– Огляд існуючих систем управління файлами на iPhone.

– Дослідження системи управління файлами на iPhone.

– Програмна реалізація системи управління файлами на iPhone.

Об'єктом дослідження є процес управління файлами на iPhone.

Предметом дослідження є методи управління файлами на iPhone.

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

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

– Удосконалено метод управління файлами на iPhone.

– Розроблено вітчизняний продукт управління файлами на iPhone, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі управління файлами на iPhone.

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 управління файлами на iPhone, 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

Призначенням системи, яка розробляється у ході виконання магістерського проектування, є управління файлами на iPhone. iPhone – лінійка чотирьохдіапазоних мультимедійних смартфонів, розроблена корпорацією Apple. Смартфони сполучають у собі функціональність плеєра iPod, комунікатора й інтернет-планшета. Працюють під керуванням операційної системи Apple iOS, що представляє собою спрощену й оптимізовану для функціонування на мобільному пристрої версію Mac OS X.

В операційній системі Apple iOS використовується ядро Darwin, що засноване на мікроядрі Mach і утримуюче код, написаний самою Apple, і код, отриманий від ОС NeXTSTEP і FreeBSD. Операційна система Apple iOS має 4 шари абстракції:

1. Ядро ОС.

2.Сервіси ядра.

3.Media.

4.API Cocoa Touch.

Розмір дистрибутива Apple iOS становить 378 МБ.

Споконвічно iOS була названа iPhone OS. Але була перейменована в iOS 7 червня 2010 року, приблизно тому, що вона тепер розраховувалася не тільки на смартфони iPhone, але й на інші пристрої Apple. 8 квітня 2010 року компанія Apple показала громадськості iOS 17.0 і запустила бета-версію 4.0 прошивання й SDK. Вони були доступні тільки для зареєстрованих розроблювачів, а остаточний варіант був випущений 21 червня 2010 року.

Більшість людей відштовхує той факт, що в iPhone, iPod touch або iPad не передбачений повноцінний файловий менеджер, недоступна робота з файлами, не можна зберігати файли на пристрої або викачувати з Інтернет.

iPhone має операційну систему, оптимізовану під невеликий екран і сенсорне керування пальцем (а також мультитач). Процес написання й налагодження програм полегшує середовище для розроблювачів – iPhone SDK. Це пакет з декількох програм, які можуть знадобитися для створення додатка на iPhone. У їхньому числі:

– Xcode – написання основного коду, налагодження, емуляція роботи iPhone для запуску й тестування програм

– Interface Builder – створення інтерфейсу програми, зв'язок між подіями інтерфейсу й кодом програми

– Instruments – аналіз ефективності використання ресурсів, навантаження на процесор. Збереження дампа пам'яті й аналіз коду в кожний момент його виконання для наступного аналізу й оптимізації

1.2 Область застосування

Областю застосування є операційна система iPhone. Операційну систему Apple iOS (до 7 червня 2010 року використовувалася назва iPhone OS) часто називають «прошиванням». На iPhone установлена модифікована версія «Mac OS X», що представляє собою оригінальну операційну систему – «Apple iOS». Аналогічна система встановлена на плеєрі iPod touch і інтернет-планшеті iPad, з тією лише різницею, що з інтерфейсу вилучені функції телефону й відправлення SMS.

Apple iOS (раніше називалася iPhone OS) – операційна система, розроблена компанією Apple на основі Mac OS X для мобільних пристроїв: iPhone, iPod Touch, iPad. Входить у сімейство операційних систем Apple OS X, до якого так само відноситься й ОС для настільних комп'ютерів – Mac OS X. В Apple

iOS використовується ядро Darwin, засноване на мікроядрі Mach і утримуюче код, написаний самою Apple, і код, отриманий від ОС NeXTSTEP і FreeBSD. iOS має чотири шари абстракції: ядро ОС, сервіси ядра, Media і API Cocoa Touch.

Одним з головних недоліків iPhone OS з моменту її появи була відсутність керування багатозадачністю з інтерфейсу користувача (однак за допомогою стороннього ПЗ, що вимагає джейлбрейка, користувач міг виконувати програми у фоновому режимі, тому що ядро iPhone OS підтримує багатозадачність). В Apple iOS версії 17 цей недолік був виправлений. Керування багатозадачністю було додано в інтерфейс користувача з використанням апаратної кнопки на передній панелі телефону для виводу списку додатків, що виконуються, і перемикання між ними. Для додатків, що працюють у фоновому режимі, виділяється найменша кількість ресурсів, завдяки чому збільшується час роботи телефону від акумулятора. Для реалізації багатозадачності в сторонніх додатках була уведена додаткова функціональність для розроблювачів, щоб створені ними додатка могли виконувати тільки «дозволені» дії у фоновому режимі.

Крім багатозадачності нова версія операційної системи одержала більше 100 нових функцій, включаючи можливість створення папок і ярликів на робочому столі, підтримку бездротової клавіатури, можливість зміни тла робочого стола (що в ранніх версіях було можливим тільки при використанні сторонніх програм). iOS 17 також забезпечує кроссплатформену підтримку всіх пристроїв від Apple і інтеграцію з iTunes. Варто відзначити, що в апаратах iPhone 14 після відновлення операційної системи до iOS 17 функціональність буде в значній мірі обмежена, так, наприклад, багатозадачність в iPhone 14 без джейлбрейка відсутній.

Поточна версія операційної системи – 17. Можливість відновлення до нової версії одержали iPhone 14, iPod Touch другого й третього поколінь. Восени операційна система стане доступної для власників iPad.

Особливості інтерфейсу

В операційній системі Apple iOS користувальницький інтерфейс Cocoa був замінений на спеціальну мобільну версію, орієнтовану на роботу пальцями з мультисенсорним екраном (без використання стилуса) – Cocoa Touch. В інтерфейсі додатка iPod застосовується технологія Cover Flow.

Інтерфейс створювався з урахуванням повного керування пальцями, тому екран не реагує на торкання іншими предметами. Технологія Multitouch, що дозволяє дисплею iPhone розпізнавати дотик до п'яти пальців одночасно, значно полегшує взаємодію користувача із пристроєм: наприклад, масштабування зображення або веб-сторінки. Випускається велика кількість адаптованих під дану технологію ігор, інтерфейс яких подібний до інтерфейсу портативної ігрової консолі PSP з перенесеними на екран елементами керування.

Є убудована можливість створення скриншотів одночасним натисканням кнопки вимикання телефону й кнопки «Додому», результат при цьому зберігається в альбомі «Фотоплівка».

Джейлбрейк

Файлова система апарата споконвічно недоступна користувачеві, через що, наприклад, відсутня можливість зміни оформлення, додавання додаткових налаштувань у меню смартфона, установки сторонніх або неліцензійних додатків. Джейлбрейк – офіційно не підтримувана Apple операція, за допомогою якої можна відкрити повний доступ до файлової системи. Після цього стає можливим установка сторонніх додатків, що заміняють iPod (наприклад, PWNPlayer) і що володіють розширеними в порівнянні зі стандартними додатками функціями; крім цього, з'являється можливість працювати з файловою системою так само, як і на звичайному ПК або КПК. На iPhone 14 за допомогою джейлбрейка реалізується підтримка багатозадачності й фонових малюнків робочого стола.

Синхронізація з Mac або PC

На даний момент єдиним офіційним засобом зв'язку iPhone з комп'ютером є програма iTunes, доступна тільки для користувачів операційних систем від

Apple і Microsoft. Створенню сторонніх програм Apple усіляко перешкоджає, аж до судового переслідування розроблювачів. Для користувачів операційних систем Linux виявляється закритим доступ не тільки до відновлень програмного забезпечення iPhone, але й до простої синхронізації смартфона з комп'ютером. Незважаючи на погрози Apple, користувачі продовжують шукати способи синхронізації iPhone з комп'ютером під керуванням операційних систем Linux. Одним зі способів синхронізації є використання бездротових мереж Wi-Fi і операційної системи Apple iOS з виконаним Джейлбрейком. Спосіб синхронізації полягає в тому, що в «розлоченій» операційній системі відкривається доступ до файлової системи смартфона, після чого комп'ютер може підключитися до нього як до звичайного накопичувача.Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи управління файлами на iPhone, 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

Розглянемо програмні продукти призначені для керування файловою системою iPhone .

iPhone Folders

Зручний файловий менеджер для роботи з файловою системою iPhone. Для входу в систему iPhone досить пройти у вікно «Мій комп'ютер», де буде відображений iPhone Folders у вигляді системного значка-провідника.

Рисунок 2.1 – Інтерфейс користувача iPhone Folders

iPhone Folders – це плагін до Провідника Windows, що дозволяє вам переглядати вміст файлової системи iPhone/iPod Touch як ще один віртуальний диск на вашому комп'ютері. Після установки ви знайдете його в папці "Мій комп'ютер" Windows 10 або "Комп'ютер" Windows 11.

Для роботи програми необхідна Windows 10/11 із встановленим iTunes, що підтримує вашу модель телефону. Для телефонів зі зробленим jailbreak файлова система відображається повністю. Для "чистих" телефонів виводиться тільки вміст папки /var/Media.

iPhone Folders реалізує всі звичні вам можливості Провідника:

– Перегляд файлової системи в різних режимах (значки, таблиця, плитка, ескізи сторінок).

– У режимі перегляду ескізів (Windows 10) і значків (Windows Vista/7) відображається вміст jpg і png файлів. Для папок, які містять додатки iPhone (.app), відображається значок додатка.

– Відкриття файлів (наприклад mp3 або txt) безпосередньо з папки телефону без попереднього копіювання їх на жорсткий диск.

– Для м'яких посилань (symlink) у режимі таблиці відображається ім'я цільового каталогу або файлу.

– Вспливаючі підказки відображають коротку інформацію з файлів, для каталогів виводиться загальний обсяг.

– Використання адресного рядка для навігації в телефоні. Для переходу до потрібної папки почніть уводити її ім'я в адресному рядку Провідника – будуть запропоновані всі підходящі варіанти.

– Пошук файлів у телефоні використовуючи стандартний пошук (Windows 10) або швидкий пошук у правом верхньому куту Провідника (Windows Vista/7).

– Копіювання файлів між телефоном і комп'ютером використовуючи контекстне меню (копіювати/вставити) або перетаскування (drag'n'drop).

– Підтримуються українські імена файлів і папок.

– При копіюванні файлів з телефону можлива автоматична конвертація файлів у форматі png із закритого формату Apple у стандартний формат, файлів у форматі plist з бінарного в xml формат.

– Створення на робочому столі (або будь-якому іншому місці на жорсткому диску) ярликів для папок у телефоні. Для створення ярлика просто перетягніть правою кнопкою папку з телефону на робочий стіл і виберіть "Створити ярлик".

– Файл прозоро для користувача копіюється в тимчасову папку й відкривається звідти.

– Після завершення додатка, використаного для відкриття файлу, він перевіряється на наявність змін і, при внесенні модифікацій, копіюється назад у телефон.

– Створення папок на телефоні.

– Перейменування й видалення файлів і папок на телефоні.

DISKAid

DiskAid – програма для комп'ютерів на базі PC і Mac, дозволяє працювати з iPhone і iPod як із зовнішніми дисками.

Запускаємо програму. Підключаємо iPhone до комп'ютера й чекаємо, поки DiskAid упізнає iphone, ipad, ipod.

За замовчуванням в DiskAid видна тільки папка /private/var/mobile/Media, але це легко виправити змінити, вибравши замість Media Folder – Root Folder (вибрати можна в лівому нижньому куті програми).

У програмі дуже простий і інтуїтивний інтерфейс. Ви можете користуватися основними меню у верху програми або ж просто перетаскувати файли з комп'ютера в потрібну папку на вашому iphone, ipod, ipad ну або ж навпаки.

Мінус у програмі, немає можливості виставляти права й міняти власника на файли, тільки копіювання/видалення.

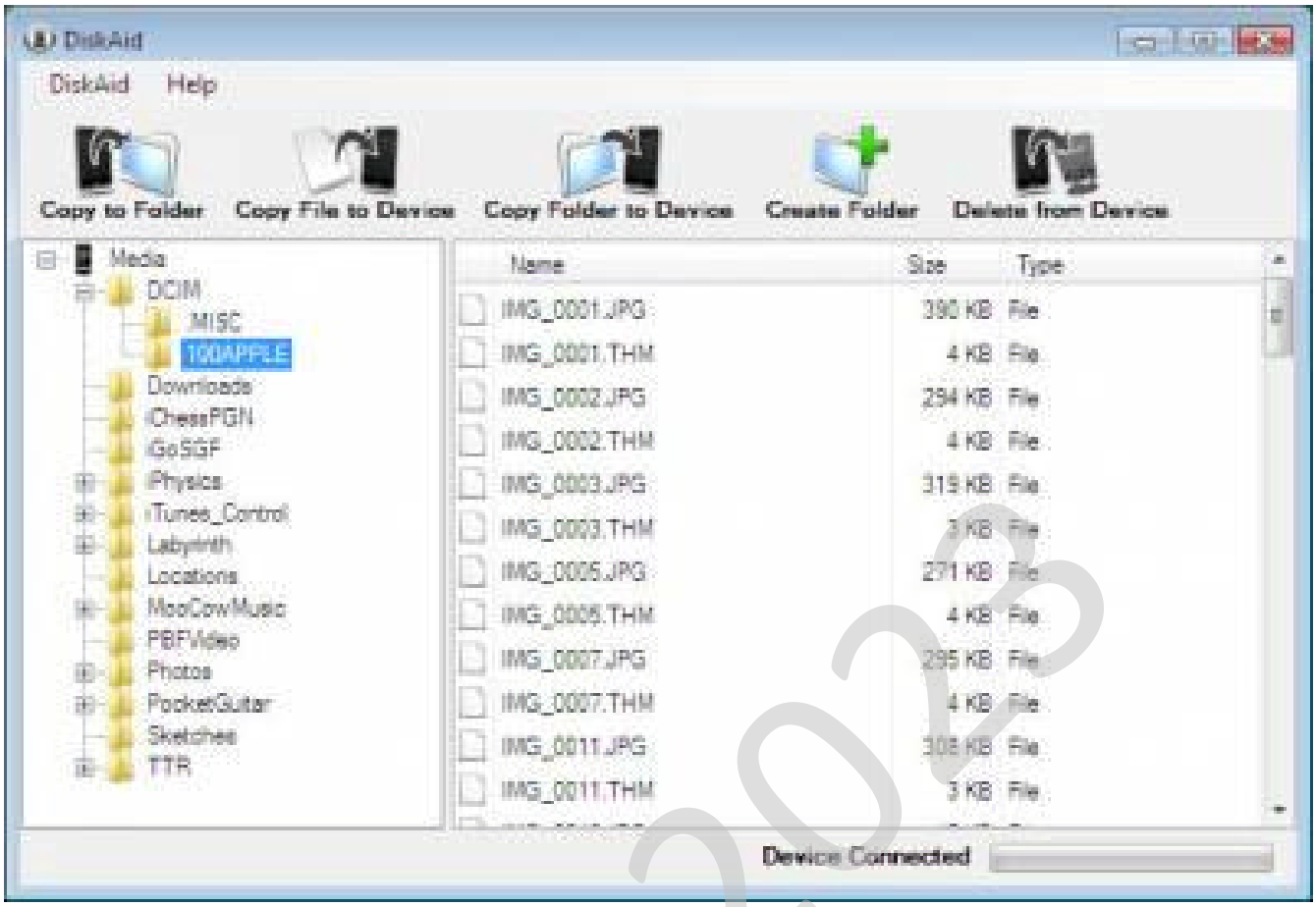


Рисунок 2.2 – Інтерфейс користувача DiskAid

За допомогою програми DiskAid Ви можете копіювати файли й папки між iPhone або iPod Touch і Вашим комп'ютером.

1. Програма DiskAid платна.

2. Програма працює як на комп'ютерах PC Windows, так і на комп'ютерах Mac OSX.

3. Програма підтримує iPhone 2G (перші покоління) і iPhone 14, а також всі моделі iPod Touch.

4. DiskAid автоматично розпізнає пристрій, досить підключити iPhone або iPod Touch до комп'ютера.

5. Дозволяє копіювати файли й папки між iPhone або iPod Touch і Вашим комп'ютером.

6. Підтримує технологію Drag & Drop (всі файли й папки можна перетаскувати мишкою).

7. Підтримує розширену роботу з файловою системою, включаючи створення папок, перейменування, копіювання й т.д.

8. Не вимагає jailbreak.

9. Підтримує всі прошивання.

iFunBox

iFunBox – це безкоштовний файловий менеджер для iPhone і iPod Touch, що працює під Windows.

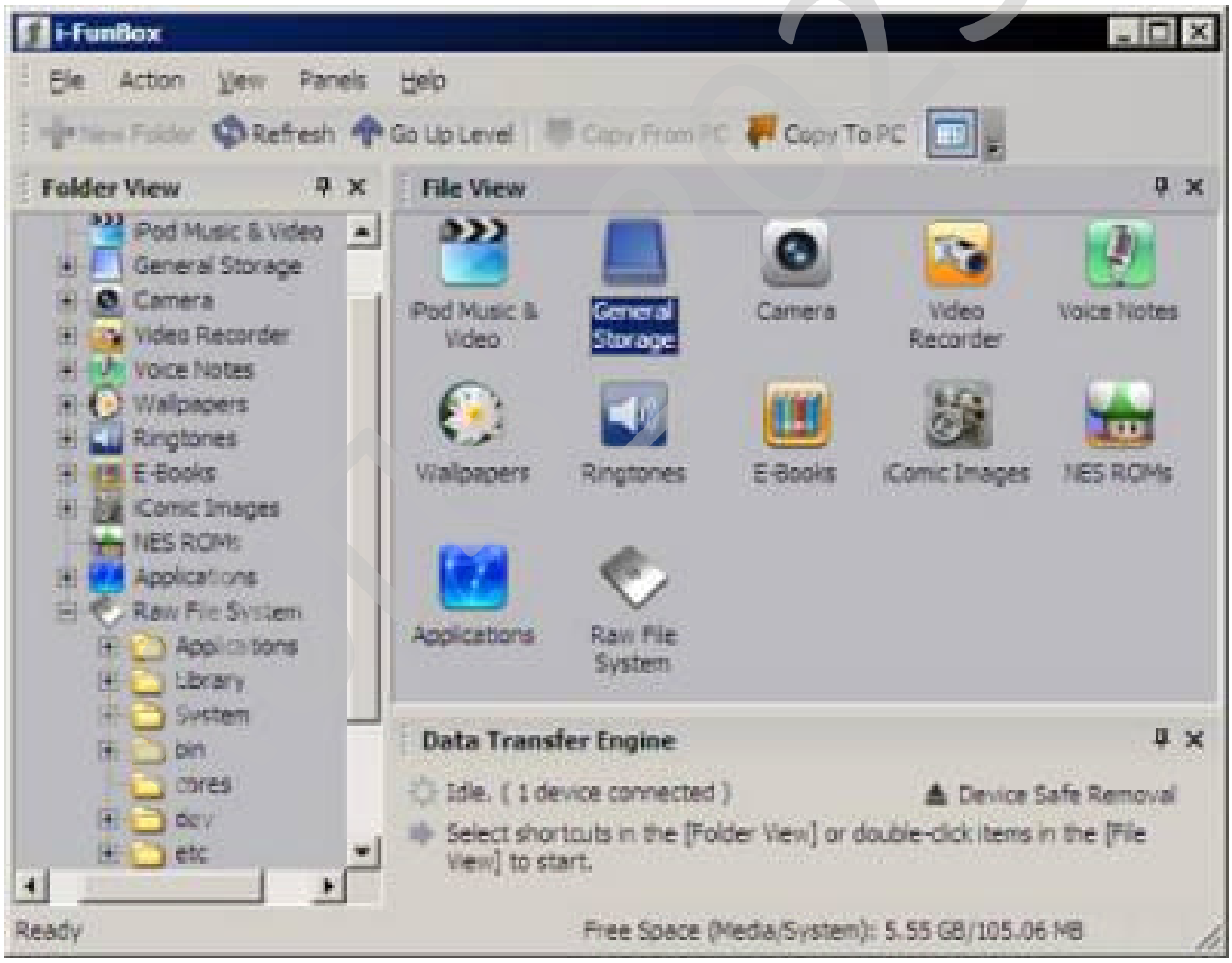


Рисунок 2.3 – Інтерфейс користувача iFunBox

Можливості програми:

1. Повне керування файловою системою iPhone і iPod (можливість копіювання файлів як на телефон, так і з його).

2. Керування файлами як в Windows E10lorer, включаючи технологію Drag & Drop, тобто файли можна звично перетаскувати мишкою.

3. Завантаження шпалер.

4. Перегляд і закачування фотографій.

5. Закачування рингтонів.

6. Висока швидкість завантаження файлів на iPhone (до 5 Мб/сек).

7. Програма не вимагає установки й запускається exe-файлом iFunBox.exe.

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 управління файлами на iPhone.

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

Надамо опис розробленої системи управління файлами на iPhone. Вона повинна:

1. Відкривати багато файлів убудованими засобами перегляду.
2. Відкривати наявні файли засобами сторонніх додатків, установлених на вашому пристрої.
3. Розроблена система управління файлами на iPhone дозволяє виконувати наступні функції:

– Створювати текстові файли.

– Записувати звукові файли.

– Імпортувати фото й відео з бібліотеки пристрою.

– Працювати з багатьма Інтернет-сервісами в тому числі Google docs, DropBox.

– Організовувати файли в каталоги/папки.

– Відправляти файли по електронній пошті.

– Відправляти документи на друк.

– Стискати дані в архів.

– Публікувати в Інтернет.

4. Додаток дозволяє завантажувати/передавати файли із ПК на пристрій і навпаки, по Wi-Fi мережі без установки додаткових програм.
5. Завантажувати із Інтернету файли за допомогою убудованого браузера.
6. Додаткові функції.

Завантаження й вивантаження файлів з iPhone, iPod touch, iPad

Запустіть додаток «Система управління файлами на iPhone» на пристрої, натисніть на іконку Wi-Fi, по середині екрана з'явиться повідомлення із

вказівкою IP адреси й номеру порту вашого iOS пристрою. Ці дані варто ввести в адресний рядок браузера на ПК. Після переходу по зазначеної IP адреси, потрапите на сторінку де буде доступний перелік файлів, що зберігаються в додатку розробленої системи управління файлами на iPhone на iOS пристрої.

Для завантаження файлів з iOS пристрою, необхідно просто нажати на файл, що цікавить вас, і він скачає браузером на ваш ПК.

Кнопка «Файли в інтернет» дозволяє завантажити файли на пристрій з вашого ПК.

Відкриття файлів убудованими засобами розробленого управління файлами на iPhone

Розроблена система управління файлами на iPhone дозволяє відкривати файли наступних форматів:

– iWorks ’08 і ’09.

– Microsoft Office – (Word, Excel, PowerPoint).

– HTML – запуск веб сторінок і веб архівів.

– PDF.

– RTF – текстові файли.

– Відео формати: MP4, MOV, MPV, 3GP, M4V.

– Аудіо формати: WAV, MP3, M4A, CAF, AIF, AIFF, ACC.

– Зображення: JPG, PNG, GIF, BMP, TIF, TIFF, ICO, CUR, XBM.

Завантаження файлів з мережі Інтернет

Для завантаження файлів з мережі Інтернет, варто скористатися убудованим браузером.

Відкриється сторінка оглядача зі стартовою сторінкою (за замовчуванням «Google»).

Знайдіть потрібний файл із мережі, натисніть на посилання й виберете потрібну вам дію: «Open» (відкрити), «Download» (скачати), «Копіювання Link»(копіювати посилання).

У випадку вибору «Download», почнеться завантаження файлу на пристрій, процес завантаження можна спостерігати нажавши на відповідну кнопку.

Відкриття файлів сторонніми додатками

Розроблена система управління файлами на iPhone уміє відкривати файли засобами сторонніх додатків.

Приклад: Вам потрібно відредагувати документ формату Exel (xls), для цього буде потрібно сторонній додаток, одним з таких є програма «Числа» (Програма для створення й редагування таблиць). Для відкриття файлу xls засобами «Числа», торкніться й утримуйте палець на імені обраного вами файлу, поки не з'явиться контекстне меню, виберіть «Відкрити у».

Розроблена система управління файлами на iPhone запропонує список додатків (з тих що встановлено на вашому пристрої) підтримуючу роботу з вашим файлом, у нашому випадку це «Числа».

Після вибору додатка, файл зкопіюється в додаток і відкриється в ньому.

Налаштування розробленої системи управління файлами на iPhone

Натисніть іконку налаштувань. З'явиться меню налаштувань, у якому доступні наступні налаштування:

«Передача файлів» – містить наступні параметри:

– Передача файлів Дозволено – вмик/вимк доступ до пристрою через Wi-Fi.

– Автентифікація – установити логін і пароль на бездротової доступ до iOS пристрою.

– Номер порту – номер порту, через який буде доступний пристрій.

– Bluetooth Передача файлів – вмик/вимк доступ через Bluetooth.

«Пароль» – дозволяє встановити пароль на запуск додатка:

– Задати – задати пароль.

– Змінити – змінити пароль.

– Блокувати – блокувати додаток при поверненні зі сну.

– Простий – простий пароль (чотири цифри), якщо виключити функцію, то можна ввести складний пароль (букви й цифри).

«Вид» – Налаштування виду файлового менеджера:

– Розширення – показувати розширення файлів.

– Передперегляд – показувати іконки передперегляду файлів.

– Іконки – показувати іконки передперегляду зі збереженням пропорцій.

«Завантаження з інтернету» – керування панеллю завантаження даних з інтернет сервісів:

– Видалення зі списку – видаляти зі списку завантажень після завершення.

– Задати папку – задати папку в яку будуть завантажуватися файли. При включенні цієї функції з'явиться поле «Вибір папки» при натисканні на яку можна буде вибрати папку з каталогу або створити нову наприклад «Download».

– Існування файлу – якщо при завантаженні, файл із такою назвою існує, то можна вибрати одне із трьох дій: «Перезапис» перезаписати на новий, «Перейменування» перейменувати, «Пропустити» пропустити й не завантажувати.

– Файли в Інтернеті – якщо при відправленні файлу на інтернет сервер, файл із такою назвою вже існує, то можна вибрати одне із трьох дій: «Перезапис» перезаписати на новий, «Перейменування» перейменувати, «Пропустити» пропустити й не завантажувати.

«Браузер» – налаштування убудованого інтернет-браузера:

– Домашня сторінка – сторінка відображувана при старті браузера.

– Остання сторінка – відкривати сторінку на якій закінчився огляд Інтернету.

– Папка завантажень – вибір місце завантаження при завантаженні файлу з інтернету: «Вибрати» – завжди запитувати куди зберігати, «Поточна Папка» – поточна папка яка була відкрита при запуску браузера, «Встановити Папку» – вибрати папку для збереження файлів за замовчуванням.

– Код – зберігає до пам’яті рядок коду й при вставці цього коду в адресний рядок Safari відкриє цю сторінку в браузері розробленої системи управління файлами на iPhone.

«Відкрити файл» – відкрити файл для перегляду.

«Pdf Переглядач» – перегляд PDF документів.

– розроблена система управління файлами на iPhone Pdf Переглядач – вмик/вимк переглядач розробленою системою управління файлами на iPhone. При включенні з'являться додаткові параметри:

– Режим Переглядача – тип перегляду: весь документ прокручуванням «Прокручування», посторінково «Посторінково» або «Перевертання» перевертання сторінок.

– Збільшення – при вмикненому режимі, щоб збільшити документ потрібно два рази кликнути одним пальцем, а щоб зменшити кликнути двічі двома пальцями.

– При вимкненому наблизити й віддалити потрібно подвійним кличем одним пальцем.

«Переглядач Зображень» – переглядач зображень:

– Затримка – час затримки перевертання слайдів.

– Повтор – повтор відтворення слайдів.

– Випадково – перегляд у випадковому порядку.

– Прихування панелі – ховати панель при перегляді зображень.

«Переглядач тексту» – редактор текстових документів:

– Тип шрифту – тип шрифту.

– Розмір шрифту – розмір шрифту.

– Колір тексту – колір тексту.

– Колір тла – колір тла.

– Відкриття редактору – завжди при відкритті файлу кликнувши по документу відкриється редактор.

– Правописання – перевірка на орфографії.

«Диктофон» – налаштування аудіо-запису на диктофон:

– дозволяє вибрати якість і кількість каналів відтворення.

«Аудіо налаштування» – налаштування відтворення звукових файлів:

– Автоматично – якщо включено, то при натисканні на трек він автоматично почне відтворення музики.

– Наступний – якщо включено, то під час відтворення, по закінченню треку почнеться наступний за списком.

– Повтор – повтор відтворення композицій по колу.

«Конвертор зображень» – формат збереження зображень із бібліотеки пристрою в середовище розробленої системи управління файлами на iPhone:

– PNG – переносить у форматі PNG.

– JPG – переносить у форматі JPG (дозволяє вибрати ступінь компресії).

«Якість відео» – якість відео з бібліотеки пристрою.

Функції розробленого управління файлами на iPhone

Розроблена система управління файлами на iPhone має стандартні функції файлового менеджера: копіювання, перенос, перейменування, видалення файлів і т.д. Для цього в списку файлів виберіть об'єкт, що цікавить вас, торкніться й утримуйте палець над файлом, поки не з'явиться спеціальне меню, у якому будуть доступні наступні функції:

– Функція «E-mail» – дозволяє оправляти файл по електронній пошті (на пристрої повинен бути настроєний поштовий клієнт).

– Функція «Передача файлів» – дозволяє передавати файли усередині Wi-Fi мережі із пристрою на пристрій або по Bluetooth, за умови що розроблена система управління файлами на iPhone установлена на обох iOS пристроях.

– Фунция «Файли в Інтернет» – дозволяє публікувати файли в Інтернет.

– Функція «Відкрити у» – дозволяє відкривати файли засобами сторонніх програм.

– Функція «Архівування» – дозволяє стискати кілька файлів в архів.

– Функція «Переміщення» – дозволяє переміщати файли з каталогу в каталог.

– Функція «Копіювання» – дозволяє копіювати файли.

– Функція «Перейменування» – дозволяє перейменувати файли.

– Функція «Видалення» – видалення файлів.

– Функція «Властивості» – властивості файлу.

Додаткові функції:

У правому нижньому куті є символ «А» відповідальний за сортування списку файлів:

– Ім’я – сортування за іменем.

– Дата – по даті.

– Розширення – по розширенню.

– Розмір – по розміру.

– Перші папки – завжди першими знаходяться папки.

– Зростання – сортувати по зростанню.

– Убування – по убуванню.

Символ «+» з лівої сторони ховає за собою наступні функції:

– Папка – створення папки.

– Текстовий файл – створення текстового файлу.

– Бібліотека – завантаження з ваших фото альбомів.

– Диктофон – запис аудіофайлу на диктофон.

«Сервіси інтернету» дозволяє прикріпити до розробленого управління файлами на iPhone інтернет сервіси:

– Flickr.

– Google Docs.

– Facebook.

– Picasa Web Albums.

– Box.Net.

– Dropbox.

– MobileMe.

– WebDav.

– FTP.

– SFTP(SSH).

– Rackspace Cloud Files.

– Amazon S3.

– Sugar Sync.

Для підключення будь-якого інтернет сервісу, необхідно просто нажати плюс у верхній лівій частині «Сервіси інтернету», вибрати потрібний вам сервіс і ввести логін і пароль.

3.2 Розробка структурної схеми

Структурна схема розробленої системи зображена на рисунку 3.1. На ній показано структуру розробленого програмного забезпечення управління файлами на iPhone. Вона складається з наступних основних структурних блоків:

– Операційна система Apple iOS.

– Система управління файлами на iPhone.

– Пам’ять.

– Комунікації.

– Вбудовані додатки.

Операційна система Apple 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.

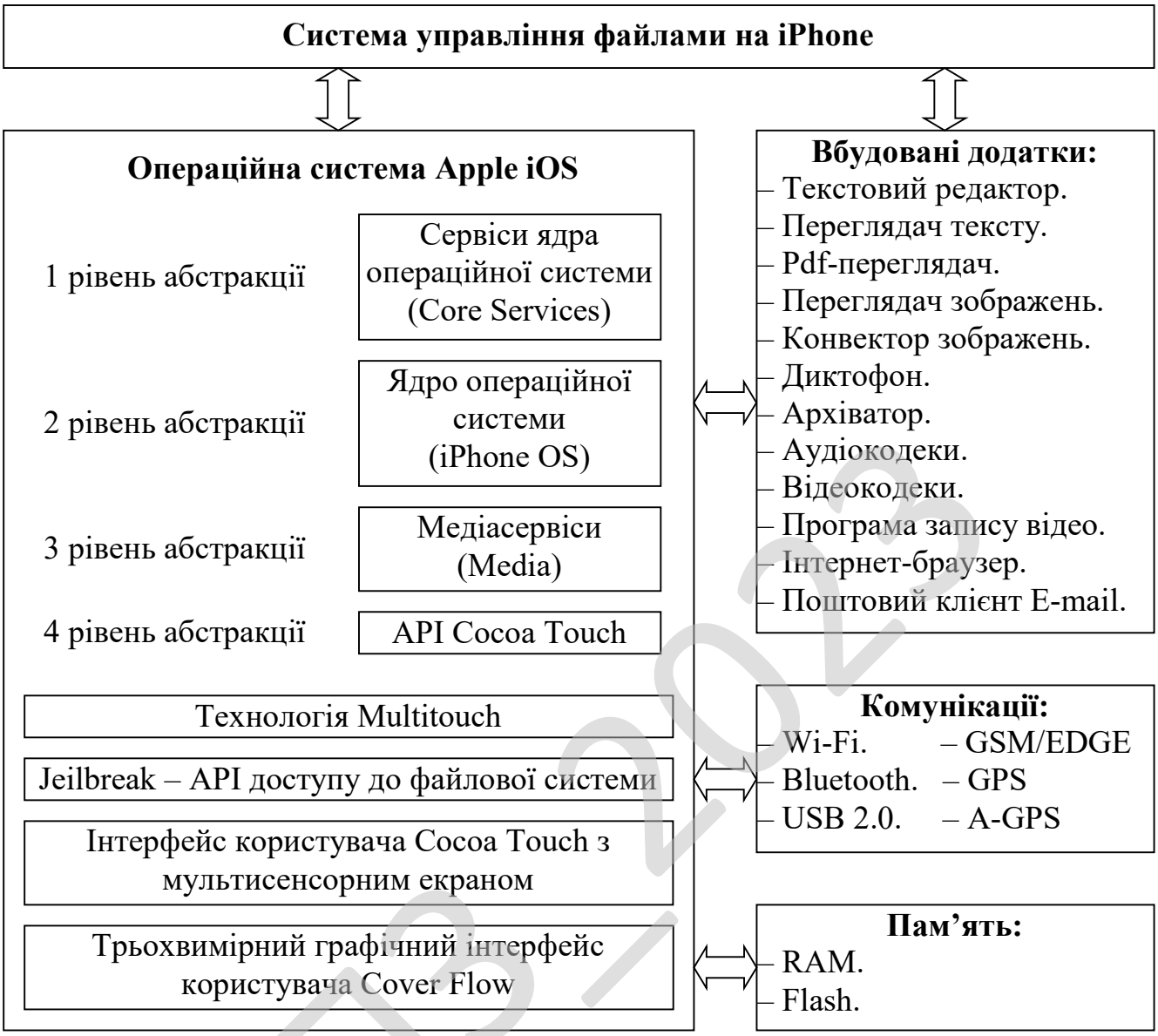


Рисунок 3.1 – Структурна схема системи

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 С і шар Cocoa мови Objective C (рисунок 3.2).

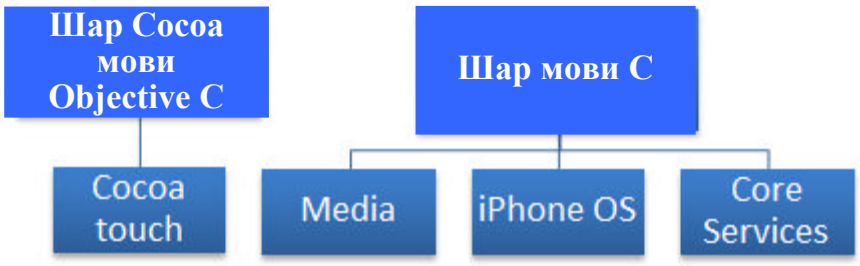


Рисунок 3.2 – Спрощена технологія iPhone

Шар мови C містить у собі шар операційної системи. Складається він з таких речей як низькорівневий файл вводу/виводу, порти й SQLite.

Пам’ять буває наступних видів:

– RAM.

– Flash.

Комунікації використовуються наступні:

– Wi-Fi.

– Bluetooth.

– USB 2.0.

– GSM/EDGE.

– GPS, A-GPS.

Jaibreak

Програма, що реалізує систему управління файлами на iPhone, написана з використанням джейлбрейка. Джейлбрейк iPhone/iPod Touch/iPad – офіційно непідтримувана Apple операція, що дозволяє відкрити програмному забезпеченню повний доступ до файлової системи пристрою. Це дозволяє розширити можливості апарата, наприклад, уможливити підтримку тим оформлення й установку додатків крім App Store. Стає можливим установити версії програвачів від сторонніх розроблювачів (наприклад PWNPlayer), що володіють розширеними можливостями в порівнянні зі стандартним, а також працювати з файловою системою, як з такий у звичайного ПК або КПК.

iOS, так само, як і Android, виконує всі додатки від імені користувача з обмеженими правами. Таким чином, додатки (незалежно від засобів розробки, використаних для їхнього створення) не можуть робити деякі системні виклики ядра ОС, а також не мають доступу до деяких директорій.

Що особливо важливо, деякі директорії стають read-only для додатків, у той час як системні процеси як і раніше зберігають можливість їхньої модифікації.

Таким чином, компанії-виробники реалізували захист від зміни важливих системних даних – ці дані можуть змінюватися тільки системним процесом, що виконує над ними строго певні операції з перевірками (наприклад, цифрових підписів на встановлюваному софті). При цьому через відсутність адмінських прав у додатків стає неможливим також і хакерські операції над цим системним процесом (за допомогою модулів ядра, ptrace і т.д.).

Обмеження встановлене, наприклад, для захисту від копіювання додатків третіх фірм.

У число «важливих системних даних» у випадку iOS входить, наприклад, колекція встановленого на телефон софта. Тобто без jailbreak неможливо занести в телефон ніякий софт, крім як з AppStore.

Jailbreak зламує наведене вище обмеження, наприклад, використанням setuid біта на виконуються файлах, що, або ослабленням прав доступу до важливих директорій.

Jailbreak для iOS дає наступні можливості:

– використання софта з боку настільного ПК (софт реалізований як надбудова над компонентами з iTunes), що дає повний доступ до файлової системи пристрою за аналогією з MS ActiveSync;

– можливість занесення в пристрій софта не з AppStore, наприклад, за допомогою Cydia. У числі цього софта є й інструменти для взлому захисту baseband (GSM-Чипа) від AT&T;

– можливість занесення в пристрій BSD-підсистеми (портування з «великого» Darwin), у яку входить SSH, тобто SSH-доступу до пристрою з можливістю виконання command-line інструментів (у тому числі інструментів для відв’язки від оператора).

Jaibreak у світі Android звичайно називається rooting, і приносить куди менше користі. Справа в тому, що Android, на відміну від iOS, не прив'язує пристрій до одному унікальному настільного ПК – для ПК пристрій виглядає так само, як його карта пам'яті, установлена в кард-рідер, і аналоги iTunes для Android не більш ніж виконують операції з файлами на цій карті. Також Android дозволяє встановлювати додатки (файли .apk) не з Google Market.

Тому rooting несе не більш ніж функцію дозволу додаткам доступу на запис до всієї файлової системи, на завантаження кернел-модулів і так далі, тобто потрібний тільки для частини додатків, і те для реалізації тільки частини можливостей даних додатків (додаток працює в неповному об'ємі й без rooting).

Існує два типи джейлбрейка:

– Прив'язаний джейлбрейк – після кожного перезавантаження iPhone потрібне підключення до комп'ютера для завантаження через модифікований бутром, на більше нових бутах телефон вимагає підключення до ПК, для відновлення ПЗ, без втрати даних.

– Повний (відв'язаний) джейлбрейк – iPhone стає незалежним від комп'ютера й може завантажуватися через нормальний завантажник з модифікованим Bootrom.

Установка додатків звичайно відбувається через менеджер пакетів, такий як Cydia. У свою чергу, він являє собою візуальний фронтенд для модифікованого менеджера пакетів Debian.

Debian – операційна система, що складається з вільного ПЗ з відкритим вихідним кодом. У цей час Debian GNU/Linux – один із самих популярних і важливих дистрибутивів Linux, у первинній формі оказавший значний вплив на розвиток цього типу ОС у цілому. Також існують проекти на основі інших ядер:

Debian GNU/Hurd, Debian GNU/kNetBSD і Debian GNU/kFreeBSD. Debian може використовуватися як операційна система для серверів, так і для робочих станцій.

Debian має найбільше серед всіх дистрибутивів сховище пакетів – готових до використання програм, – і якщо навіть не по їхньому числу, то по числу підтримуваних архітектур: починаючи з ARM, використовуваної у вбудовуються устройствах, що, найбільш популярних x86 і PowerPC, нових 64-розрядних AMD, і закінчуючи IBM S/390, використовуваної в мейнфреймах. Для роботи зі сховищем розроблені різні засоби, саме популярне з яких – Advanced Packaging Tool (APT).

Debian став основою цілого ряду дистрибутивів (більше 100). Найвідоміші з них – Adamantix, Bioknoppix, Dreamlinux, Clusterix, Gnoppix, Knoppix, Ubuntu, Libranet, Linspire, MEPIS, Xandros Desktop OS і Maemo.

На Debian заснована безліч дистрибутивів, у тому числі Ubuntu, MEPIS, Dreamlinux, Damn Small Linux, Xandros, Knoppix, Linspire, aptosid, Kanotix, Parsix, LinEx, Linux Mint і інші.

Debian відрізняється багатством можливостей. У поточну стабільну версію включено понад двадцять дев'ять тисяч пакетів програм для дев'яти архітектур на основі ядра Linux (від Intel/AMD 32-bit/64-bit, широко застосовуваних у персональних комп'ютерах, до ARM, звичайно використовуваних у вбудовуються системах, що, і мейнфреймах IBM System z) і також двох архітектур на основі ядра FreeBSD (kfreebsd-i386 and kfreebsd-amd64).

Відмітними рисами Debian є система керування пакетами Advanced Packaging Tool (APT), тверда політика стосовно пакетів, репозиторії з величезною їхньою кількістю, а також висока якість версій, що випускаються. Це уможливило просте відновлення між версіями, а також автоматичну установку й видалення пакетів.

При стандартній установці Debian використовується середовище робочого стола GNOME, куди включений набір популярних програм, таких як OpenOffice.org, Iceweasel (модифікація Firefox), поштова програма Evolution,

програми для запису CD/DVD, програвачі музики й відео, програми для перегляду й редагування зображень і програми для перегляду документів у форматі PDF. Також є образи CD, зібрані з KDE, Xfce і LXDE. Установні диски (у кількості п'яти (Lenny) або восьми (Squeeze) DVD або більше тридцяти CD) містять всі доступні й не обов'язково необхідні для стандартної установки пакети. Для методу установки по мережі використовується CD, що менше звичайного настановного CD/DVD. Він містить тільки те, що необхідно для запуску установника й завантаження пакетів, обраних у процесі установки за допомогою APT. Образи CD/DVD можна вільно скачати через BitTorrent, jigdo або купити в одного з постачальників компакт-дисків Debian.

Саме в Дебіані вперше був уведений як єдиний стандарт механізм вибору кращого ПЗ серед декількох варіантів – Alternatives.

Релізи Debian розділені на три гілки:

– стабільну (stable), що містить пакети, що ввійшли в останній офіційний дистрибутив (відновлення пакетів у ньому відбувається тільки для усунення уразливостей);

– тестуєму (testing), з якої буде формуватися наступний стабільний дистрибутив;

– нестабільну (unstable), у якій пакети готуються до приміщення в тестуєму гілку.

Існує також гілка, називана експериментальної (e10erimental); у неї містяться пакети, що перетерплюють особливо більші зміни.

Пакети програмного забезпечення (ПЗ), які перебувають у розробці, споконвічно попадають або в дистрибутив проекту за назвою unstable (також відомий як sid), або в репозиторій e10erimental. Версії ПЗ в unstable досить стабільні, щоб побачити світло на думку їхніх розроблювачів, але вони вже містять специфічні зміни, внесені в рамках проекту Debian, у тому числі інформацію для створення пакета дистрибутива. Ці зміни можуть бути новими й не тестованими. ПЗ, не готове до випуску, звичайно розміщається в e10erimental.

Після того, як версія ПЗ пробуде в unstable деяку кількість часу (залежно від критичності змін), пакет автоматично переходить у дистрибутив testing. Перехід відбувається тільки якщо пакет не містить критичних помилок, і всі інші необхідні пакети задовольняють умовам переходу в testing.

Так як відновлення пакетів Debian між офіційними релізами не містять у собі нову функціональність, деякі розроблювачі використовують пакети з unstable і e10erimental для нових версій своїх продуктів. Однак ці дистрибутиви тестуються не так ретельно, як stable і можуть не одержувати вчасно важливі виправлення безпеки. Іноді неуважне відновлення на працюючі пакети з unstable може серйозно порушити роботу ПЗ.

Після того, як пакети в testing дозріли, і мети, поставлені перед наступним релізом, досягнуті, testing стає наступною стабільною версією. Останній стабільний реліз Debian за назвою «Squeeze» одержав порядковий номер 6.0 6-го лютого 2011. Наступний реліз 7.0 має кодове ім'я «Wheezy».

Розглянувши Debian повернемося до Jailbreak. Якщо буде потреба можливо повернути оригінальний заводський стан за допомогою iTunes. Так як Jailbreak сугубо програмна операція, то при цьому всі її сліди, як прийнято вважати, губляться. Теоретично можливе ведення журналу прошивання в спеціальному розділі пам'яті апарата, але підтвердження або спростування даної гіпотези важко, через закритість вихідного коду системи. Правда, існує гіпотетична погроза відмови блоку, відповідального за кабельну сполуку пристрою з комп'ютером. При цьому заводське прошивання буде не повернути до здачі в ремонт, що приведе до втрати гарантії. Однак, таких випадків дотепер не відбувалося.

З одного боку, джейлбрейк корисний, так як дозволяє трохи компенсувати користувачам апаратів незручності, пов'язані із пропрієтарною ліцензією Apple, як то: незручність користування App Store, прив'язка до одного комп'ютера й т.д. Також розблокований апарат здобуває функції, що давно стали стандартом де-факто в інших коммунікаторах/КПК і були відсутні в iPhone/iPod Touch,

наприклад, вільний доступ до файлової системи (реалізується за допомогою iFile або подібного софта з Cydia), або відправлення MMS (відсутній в iPhone 2G).

З іншого боку, jailbreak може принести незручності, так як програмне забезпечення, доступне в Cydia, на відміну від App Store, не перевіряється на надійність і безпеку й може завдати шкоди даним на апарату або стати причиною їхньої крадіжки. Так, установка пакета OpenSSH (у цілому дуже корисного), що відбувається автоматично при виконанні процедури джейлбрейка, без зміни паролів UNIX на апарату – застава викрадення даних, що для ділових людей може завдати значної шкоди.

Відомі випадки злому телефонів, підданих джейлбрейку. Відомі випадки, коли користувачі пристроїв з Jailbreak не могли одержати доступ до каталогу електронних книг iBooks.

Процедури злому, аналогічні Jailbreak для iOS, існують і для інших мобільних платформ із обмеженнями на установку додатків і доступ до файлової системи, включаючи Android і Symbian 9 і для ігрових консолей PS3.

Xcode

Крім того, програма, що реалізує систему управління файлами на iPhone, написана з використанням Xcode.

Xcode – інструментарій розробки додатків під Mac OS X і Apple iOS, розроблений компанією Apple. Поставляється безкоштовно на дистрибутивному диску Mac OS X Install DVD разом з операційною системою Mac OS X, але не встановлюється за замовчуванням. Остання версія – 4.0, не підтримується старими версіями Mac OS. Відновлення можна завантажити безкоштовно на офіційному сайті підтримки.

Основним додатком пакета є убудоване середовище розробки, що називається Xcode. Крім цього, пакет Xcode містить у собі більшу частину документації розроблювача від Apple і Interface Builder – додаток, що використовується для створення графічних інтерфейсів.

Пакет Xcode містить у собі змінену версію вільного набору компіляторів GNU Compiler Collection (GCC, apple-darwin9-gcc-4.0.1) і підтримує мови C, C++, Objective-C, Objective-C++, Java, AppleScript, Python і Ruby з різними моделями програмування, включаючи (але не обмежуючись) Cocoa, Carbon і Java. Сторонніми розроблювачами реалізована підтримка GNU Pascal, Free Pascal, Ada, C#, Perl, Haskell і D. Пакет XCode використовує GDB у якості back-end'a для свого відладника.

У серпні 2006 Apple оголосила про те, що DTrace, фреймворк динамічного трасування від Sun Microsystems, випущений як частина OpenSolaris, буде інтегрований в Xcode за назвою Xray. Пізніше Xray був перейменований в Instruments.

З 9 березня 2011 року стала доступна нова версія Xcode 4, причому в нову версію входять набори SDK OS X 10.6 і SDK iOS 17.3. Уперше попередній реліз Xcode 4 був показаний на WWDC 2010.

В Xcode 4 програмісти побачать новий користувальницький інтерфейс: єдине вікно, куди убудоване й Interface Builder, і сам Xcode, і Instruments; наявність помічника; нову систему аналізу коду Debug Console з більше сильним «движком», що полегшує виправлення помилок і здійснює пошук логічних проблем у коді. Програма також одержала додатковий компілятор LLVM.

Cocoa Touch

Технологія Cocoa на iPhone називається Cocoa Touch (вона така ж як і звичайне Cocoa), так як iOS складається з подій дотику (touch). Коли ви доторкаєтеся до екрана iPhone (tap) – відбувається подія touch. Події touch дозволяють нам програмувати події на дотики користувачів.

Cocoa Touch супроводжується бібліотеками класів, необхідних для розробки додатка на iPhone. При розробці додатка на iPhone використовуються два framework – це Foundation framework і UIKit framework. Framework – це колекція кодів, що вирішують аналогічні завдання. Foundation framework присвячена стандартним темам програмування таким, як колекції,

рядки, файли вводу/виводу й інші базові завдання. UIKit присвячена інтерфейсу iPhone і містить такі класи як UIview. При вивченні більше часу ми будемо приділяти UIKit.

Foundation Framework

Foundation framework містить класи мови Objective C**,** які обгортають усередині себе функції низькорівневого програмування. Наприклад, замість того, щоб працювати з низькорівневими файлами вводу/виводу можна працювати із класом NSFileManager. Foundation framework супроводжується безліччю класів, які реально повинні бути вивчені, якщо ви хочете розробляти додатка для iPhone**.**

iPhone Frameworks

Нижче представлений список framework, які доступні розроблювачеві iPhone:

– Framework – Призначення.

– AddressBook – Доступ до списку контактів користувача.

– AddressBookUI – Відображення списку контактів.

– AudioToolbox – Потоки аудіоданих; запис і програвання відео.

– AudioUnit – Аудіо одиниці.

– CFNetwork – Стільниковий і wi-fi інтернет.

– CoreFoundation – Схожий на Foundation framework, але рівнем нижче (краще його не використовувати).

– CoreGraphics – Quartz 2D.

– CoreLocation – Місце розташування користувача/GPS.

– Foundation – Шар Cocoa foundation.

– MediaPlayer – Програвання відео.

– OpenAL – Позиційні аудіобібліотеки.

– QuartzCore – Анімація.

– Security – Сертифіковані ключі й довірча політика.

– SystemConfiguration – Конфігурація інтернет.

– UIKit – Користувальницький інтерфейс iPhone.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.3. З рисунку видно, що розроблена система складається з наступних частин:

– Система управління файлами на iPhone.

– Параметри системи управління файлами на iPhone.

– Блок завантаження/вивантаження файлів.

– Відкриття файлів.

Розглянемо більш детально кожний з блоків розробленої програми.

Система управління файлами на iPhone 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.

Параметри файлового менеджера складаються з наступних функціональних блоків:

– Пароль.

– Вид.

– Налаштування інтернет-браузера.

– Аудіо налаштування.

– Конвертор зображень.

– Якість відео.

– Налаштування передачі файлів.

– Налаштування редактору текстових документів.

Рисунок 3.3 – Функціональна схема системи

Блок завантаження/вивантаження файлів складається з наступних функціональних блоків:

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

– Завантаження файлів з мережі Інтернет.

Блок відкриття файлів:

– Убудованими засобами розробленої системи управління файлами на iPhone.

– Відкриття файлів сторонніми додатками.

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 3.4. З нього видно, що процеси взаємодіють наступним чином. Першим процесом, який завантажується у системі є процес виведення головного вікна додатку.

Цей процес взаємодіє з наступними процесами:

– Процес завантаження файлів із зовнішнього пристрою.

– Процес виведення списку накопичувачів.

Процес завантаження файлів із зовнішнього пристрою взаємодіє з наступними процесами:

– Процес підключення до зовнішнього пристрою.

– Процес вибору файлу для завантаження.

– Процес вибору накопичувача для запису.

Процес виведення списку накопичувачів взаємодіє з процесом вибору накопичувача.

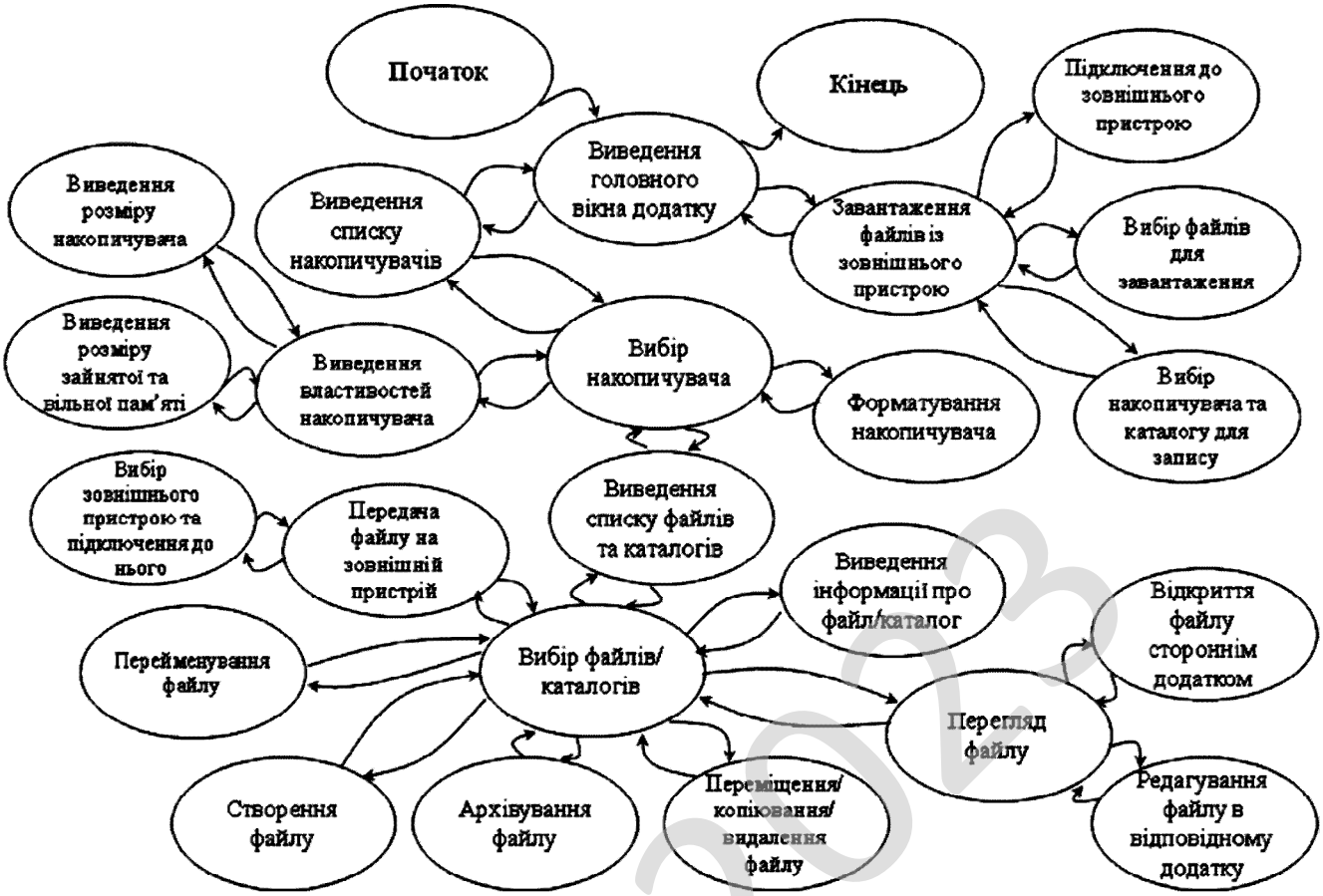


Рисунок 3.4 – Діаграма взаємодії процесів

Процес вибору накопичувача взаємодіє з наступними процесами:

– Процес форматування накопичувача.

– Процес виведення властивостей накопичувача.

– Процес виведення списку файлів та каталогів.

Процес виведення властивостей накопичувача взаємодіє з наступними процесами:

– Процес виведення розміру накопичувача.

– Процес виведення розміру зайнятої та вільної пам'яті.

Процес виведення списку файлів та каталогів взаємодіє з процесом вибору файлів та каталогів.

Процес вибору файлів та каталогів взаємодіє з наступними процесами:

– Процес передачі файлів на зовнішній пристрій, який, у свою чергу взаємодіє з процесом вибору зовнішнього пристрою та підключенням до нього.

– Процес перейменування файлу.

– Процес створення файлу.

– Процес архівування файлу.

– Процес переміщення/копіювання файлу.

– Процес виведення інформації про файл/каталог.

– Процес перегляду файлу.

Останній процес взаємодіє з наступними процесами:

– Процес відкриття файлу стороннім додатком.

– Процес редагування файлу в відповідному додатку.

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

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

4.1 Блок–схеми та опис алгоритмів функціонування системи

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

Спершу відбувається виведення основного вікна програми. Після цього відбувається виведення списку користувачів.

Наступним кроком є обирання накопичувача.

За цією дією виводиться поточний стан каталогів та властивостей накопичувача.

Після цього користувач обирає одну з наступних дій:

– Зміна накопичувача.

– Перехід в інший каталог.

– Обирання файлу.

– Перегляд файлу.

– Створення/редагування файлу.

– Копіювання/переміщення файлу.

– Видалення файлу.

Якщо користувач обирає зміну накопичувача, то відбувається зміна накопичувача на обраний.

Якщо користувач обирає перехід у інший каталог, тоді відбувається виконання наступних дій:

– Перехід у вказаний каталог.

– Виведення змісту вказаного каталогу.

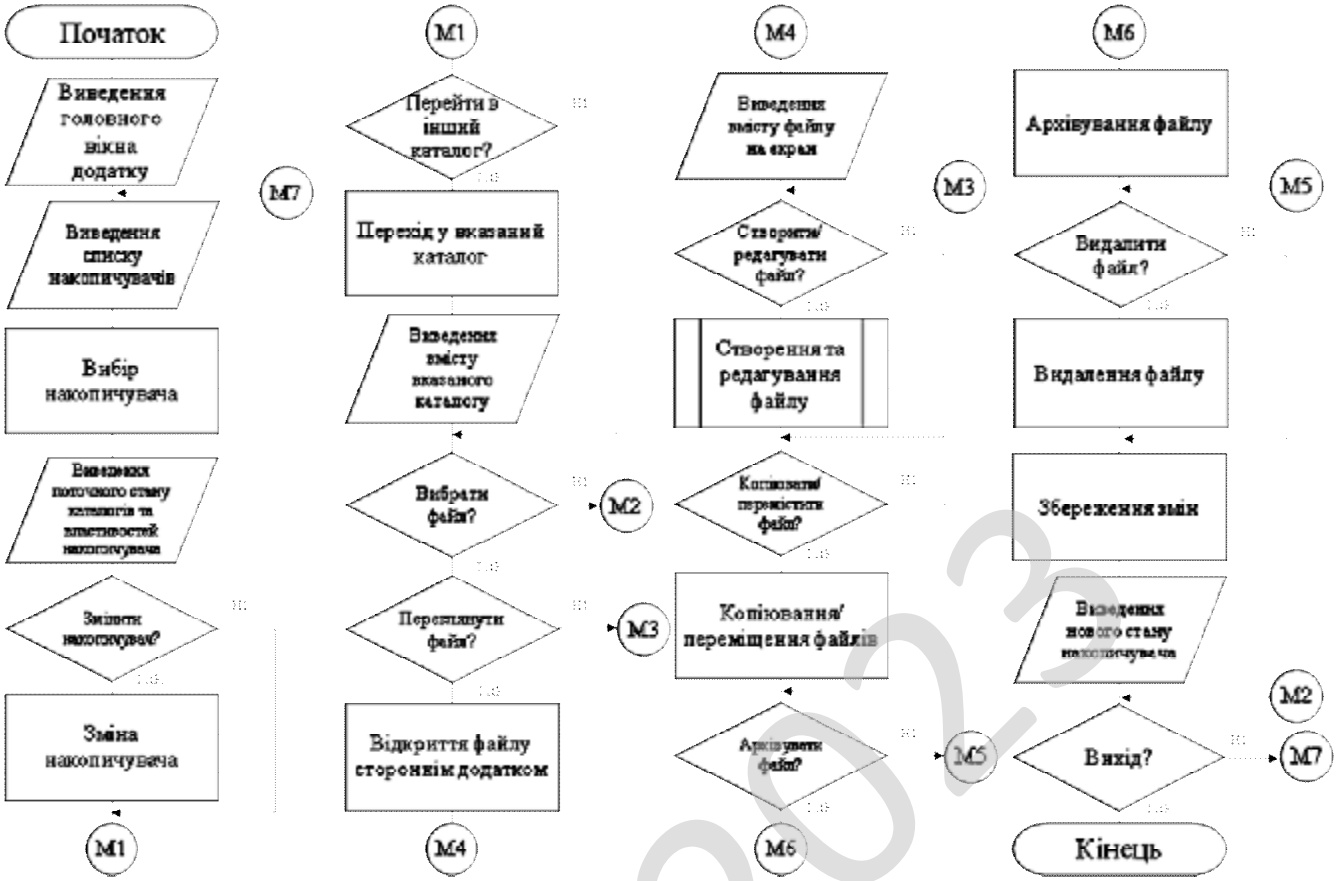


Рисунок 4.1 –Блок-схема основної програми

Розглянемо, які операції може виконувати користувач з файлами, які знаходяться на iPhone.

Якщо користувач обирає перегляд файлу, тоді виконуються наступні дії:

– Відкривається файл стороннім додатком.

– Виводиться зміст файлу на екран.

Якщо користувач обирає створення/редагування файлу, тоді виконуються наступні дії:

– Запускається підпрограма створення/редагування файлу.

Якщо користувач обирає архівування файлу, тоді виконуються наступні дії:

– Запускається підпрограма архівування файлу.

Якщо користувач обирає видалення файлу, тоді виконуються наступні дії:

– Запускається підпрограма видалення файлу.

Після виконання усіх вищевизначених операцій над файлами та файловою системою відбуваються наступні дії:

– Зберігаються зміни.

– Виводиться новий стан накопичувача.

Після цього користувач визначає, працювати йому далі з програмою, або ні.

На рисунку 4.2 зображено блок-схему алгоритму роботи підпрограми створення та редагування файлу. Вона працює наступним чином.

Користувач може обрати одну з наступних дій:

– Створення/редагування текстового файлу.

– Створення аудіофайлу.

– Створення відеофайлу.

– Створення файлу зображення.

– Редагування файлу зображення.

– Створення електронного листа.

Якщо користувач обирає створення/редагування текстового файлу, тоді відбувається запуск текстового редактору.

Якщо користувач обирає створення аудіофайлу, тоді відбувається запуск диктофону.

Якщо користувач обирає створення відеофайлу, тоді відбувається запуск програми запису відео.

Якщо користувач обирає створення файлу зображення, тоді відбувається запуск програми створення фото.

Якщо користувач обирає редагування файлу зображення, тоді відбувається запуск графічного редактора.

Якщо користувач обирає створення електронного листа, тоді відбувається запуск поштового клієнта.

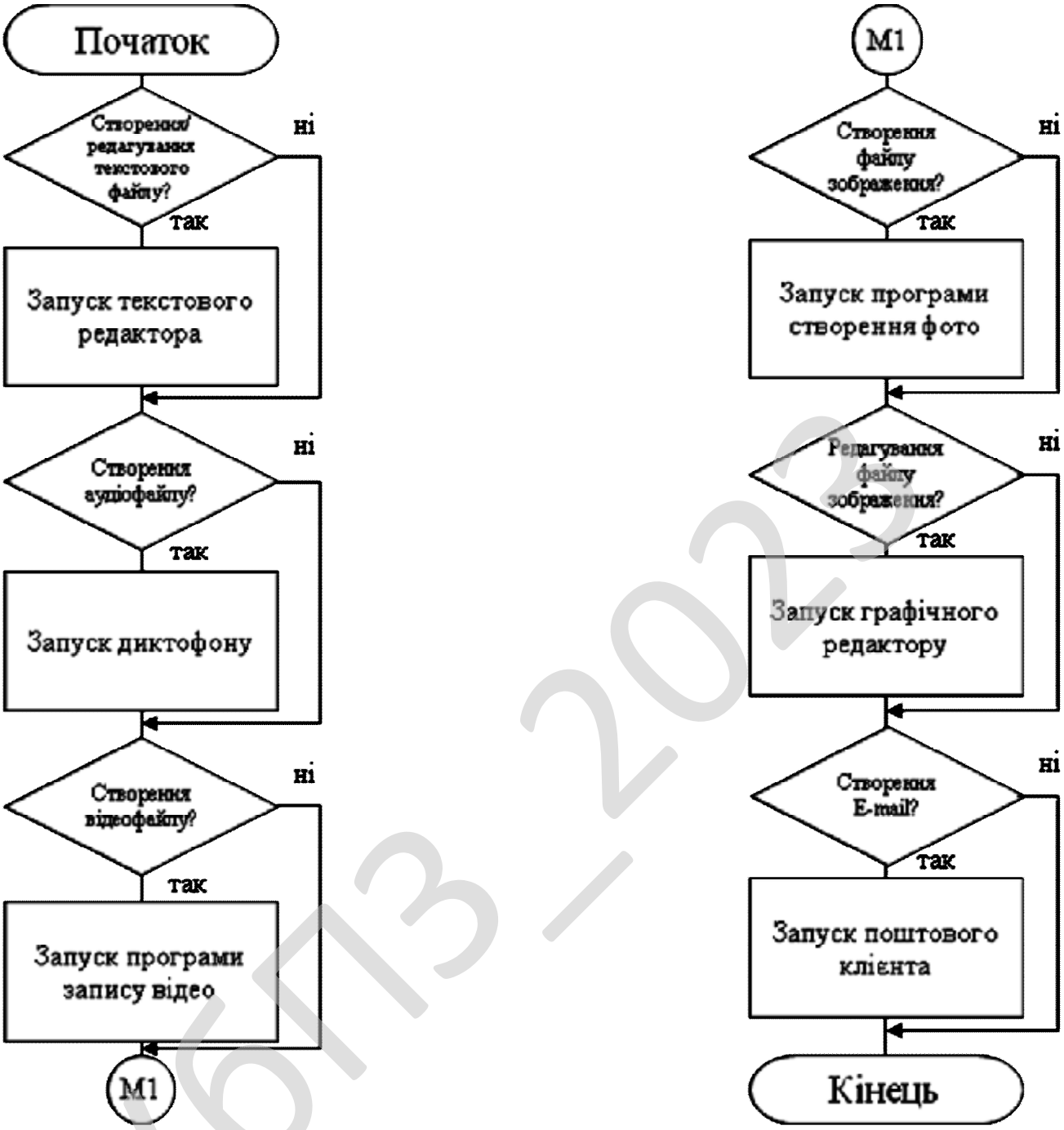


Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми створення та редагування файлу

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

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).

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.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму UMAC (код автентифікації повідомлення на основі універсального гешування) – один з видів коду автентичності повідомлень (MAC).

Швидка «універсальна» функція використовується, для того, щоб гешувати вхідне повідомлення M у короткий рядок. До цього рядка потім застосовується функція XOR із псевдовипадковим значенням, у результаті чого ми одержуємо тег UMAC:

$$Tag=H_{K1}(M)\oplus F_{K2}(Nonce)$$

де K1 і K2 – секретні випадкові ключі, які мають одержувач і відправник.

Звідси видне, що безпека UMAC залежить від того, яким випадковим способом відправник і одержувач вибрали таємну геш-функцію й псевдовипадкову послідовність. При цьому значення Nonce міняється кожний такт. Через використання Nonce, приймач і передавач повинні знати час відправлення повідомлення й принцип створення значення Nonce. Замість цього

можна використовувати в якості Nonce будь-яке інше неповторюване значення, наприклад порядковий номер повідомлення. При цьому даний номер не зобов'язано бути секретним, головне щоб він не повторювався.

UMAC розрахований на використання 32-х, 64-х, 92-х, і 128-бітових тегів, залежно від необхідного рівня безпеки. UMAC звичайно використовується разом з алгоритмом шифрування AES.

Функція створення ключа й псевдовипадкової послідовності

Створення псевдовипадкових байтів необхідно для роботи UHASH і при створенні тегів

Вибір блокового шифру

Для своєї роботи UMAC використовує блоковий шифр, вибір якого визначають наступні константи:

– BLOCLEN – довжина, у байтах, блоку з яким працює блоковий шифр.

– KEYLEN – довжина, у байтах, ключа блокового шифру.

При цьому використовується функція

– ENCIPHER(K,P) – зашифрувати рядок P з BLOCLEN байтів, використовуючи ключ K.

Приклад: якщо використовується AES з 16-байтним ключем, то BLOCLEN буде рівним 16(тому що AES працює з 16-байтними блоками).

KDF – функція створення ключа

Ця функція генерує послідовність псевдовипадкових байтів, використовуваних для ключових геш-функцій.

Вхід:

– K – рядок довжиною KEYLEN байт. / / Ключ блокового шифру.

– Index – ненегативне ціле число менше, чим 2 ^ 64.

– Numbytes – ненегативне ціле число менше, чим 2 ^ 64.

Вихід:

– Y – рядок довжини numbytes байт.

PDF: функція створення псевдовипадкового числа

Ця функція ухвалює ключ і даний час і повертає псевдовипадкове число для використання його в тегу покоління. За допомогою цієї функції можуть бути отримані числа довжиною 4, 8, 12 або 16 байт.

Вхід:

– K – рядок довжиною KEYLEN байт.

– Nonce – рядок довжиною від 1 до BLOCKLEN байт.

– Taglen – ціле число 4, 8, 12 або 16.

Вихід:

– Y – послідовність байтів довжини taglen.

Генерація UMAC-тегів

Генерація UMAC-тегів відбувається за допомогою UHASH функції при використанні Nonce значенні й отриманої до цього рядка. Їхня довжина може бути 4, 8, 12 або 16 байт.

Вхід:

– K – рядок довжиною KEYLEN байт.

– M – рядок довжиною менше 267 біт.

– Nonce – випадкове число від 1 до BLOCKLEN байт.

– Taglen – ціле 4, 8, 12 або 16.

Вихід:

– Тег, послідовність байтів довжиною taglen.

Алгоритм обчислення тегів:

Hashedmassage = UHASH(K, M, Taglen)

Pad = PDF(K, nonce, Taglen)

Tag = Pad xor Hashedmassage

UMAC-32 UMAC-64 UMAC-96 UMAC-128

Дані позначення містять у своїй назві певне значення довжини тегу:

– UMAC-32 (K, M, Nonce) = UMAC (K, M, Nonce, 4).

– UMAC-64 (K, M, Nonce) = UMAC (K, M, Nonce, 8).

– UMAC-96 (K, M, Nonce) = UMAC (K, M, Nonce, 12).

– UMAC-128 (K, M, Nonce) = UMAC (K, M, Nonce, 16).

Універсальна функція гешування(UHASH)

UHASH – універсальна функція гешування, серцевина алгоритму UMAC. UHASH – функція працює в три етапи. Спочатку до вхідного повідомлення застосовується L1-HASH, потім до цього результату застосовується L2-HASH і, нарешті, до результату застосовується L3-HASH . Якщо при цьому довжина вхідного повідомлення не більш 1024 біт, то L2-HASH не використовується. Тому що функція L3-hash повертає тільки слово довжини 4 байта, те якщо потрібно одержати геш довжини більше 4 байт, здійснюється кілька ітерацій даної трирівневої схеми.

Універсальна функція

Нехай функція гешування вибирається із класу геш-функцій H, які відображають повідомлення в D, набір усіляких образів повідомлення. Цей клас називається універсальним, якщо для яких-небудь окремих пар повідомлень, існує на безлічі H/D функцій, функція, яка відображає їх в елемент D. Зміст цієї функції в тому, що якщо третя сторона прагне замінити одне повідомлення іншим, але при цьому вважає, що геш-функція була обрана абсолютно випадково, те ймовірність не виявлення підміни стороною, що ухвалює, прагне до 1/D.

L1-hash – перший етап

L1-hash розбиває повідомлення на шматки з 1024 байт і до кожного шматка застосовує алгоритм гешування називаний NH. Вихідний результат алгоритму NH в 128 раз менше вхідного.

L2-hash – другий етап

L2-hash працює з виходом L1-hash, використовує поліноміальний алгоритм POLY. Другий етап гешування використовується, тільки якщо довжина вхідного повідомлення більше 16 мегабайт. Використання алгоритму POLY потрібно для того, щоб уникнути тимчасову атаку. На виході з алгоритму POLY виходить 16 байтне число.

L3-hash – третій етап

Цей етап потрібно для того щоб з вихідних 16 байтів алгоритму L2-hash одержати 4-байтне значення.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено головне вікно програми, з якого видно, що програма складається з наступних блоків:

– Блок визначення носія інформації.

– Блок визначення елемента системи.

– Блок визначення можливостей роботи з обраним елементом файлової системи.

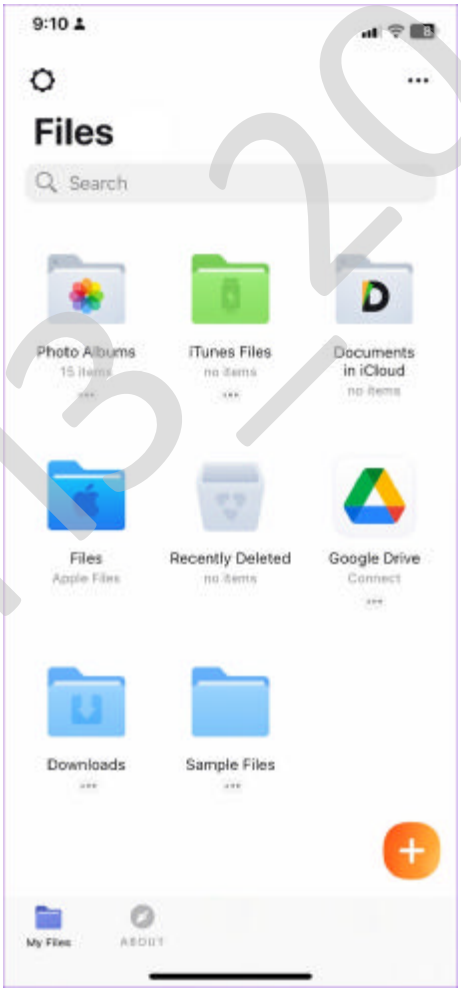


Рисунок 5.1 – Головне вікно програми

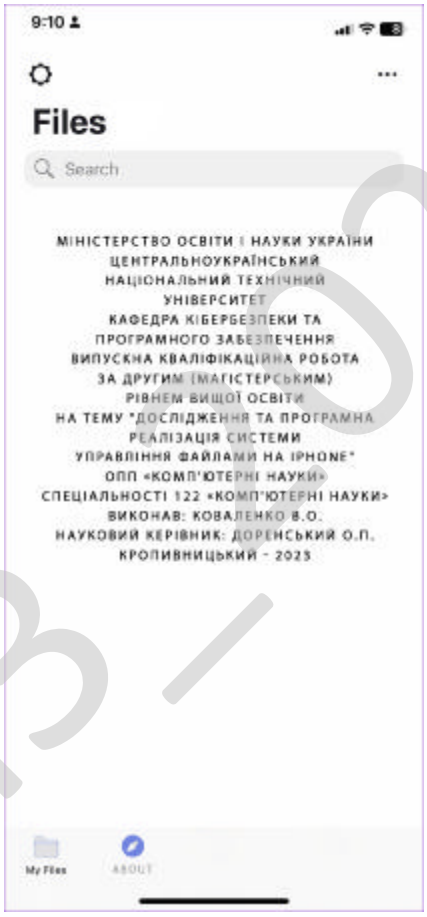
На рисунку 5.2 зображено вікно довідки про програму, з якої є можливість отримати наступну інформацію:

– Тема проекту.

– Виконавець проекту.

– Керівник проекту.

– Місце виконання проекту.



t

Рисунок 5.2 – Довідка

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи управління файлами на iPhone.

Метою розробки є дослідження та програмна реалізація системи управління файлами на iPhone.

Об'єктом дослідження є процес управління файлами на iPhone.

Предметом дослідження є методи управління файлами на iPhone.

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

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

– Удосконалено метод управління файлами на iPhone.

– Розроблено вітчизняний продукт управління файлами на iPhone, 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 48 днів (два місяці).

В магістерській роботі було проведене дослідження та розроблене програмне забезпечення системи управління файлами на iPhone.

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	130
3. Запланований термін розробки, днів	Fpq	48 (2 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
7. Кількість макетів вхідної інформації	–	3

Продовження табл. 7.1

1	2	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 табл. 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 ПО для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн	–	13000
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	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

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

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

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

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

$$B=1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}026$$

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

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

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

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

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

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

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

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

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

де С – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0{,}3\cdot 2{,}66\cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)}\cdot 42=71 \text{ люд/день}$$

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	71	Ф 7.1-7.4
Впровадження	13	Д13
Всього	112	–

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,

$$Ч = \frac{112 \cdot 1}{48\text{-}5} = 2,6 \text{ ставки}$$

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	9	810	13,5
Монітор	60	9	540	9
Клавіатура	30	9	270	4,5
Маніпулятор «мишка»	30	9	270	4,5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	2	240	4
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор-маршрутизатор	30	3	90	1,5
Кабельні господарства ЛОМ на 1 м. п.	2,5	320	800	13,33
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	53,99

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

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

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

$$Ф_{др}^{с} = \frac{54 \cdot 3}{1,2} = 135 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

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

$Ч_{ел}$=135/(60·8)=0,28 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	Кількість штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server, серверу доступу АДСЛ (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	0,2	0,1
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,2	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,2	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	0,2	
Всього		0,8	

Продовження табл. 7.4

Посада	Вид роботи	Час	Кількість штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,4
	Підтримка постійних клієнтів	1	
	Оформлення договорів, ведення тендерів	1	
	Контроль взаєморозрахунків з постачальниками	0,2	
Всього		3,2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	0,2	0,1
	Створення графічних і стилістичних елементів сайту	0,2	
	Оформлення банерів і промо-сторінок	0,2	
	Розміщення графіки і контенту на Інтернет сторінках	0,2	
Всього		0,8	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	0,2	0,1
	Верстка друкованих видань	0,2	
	Додрукова підготовка макетів	0,2	
	Розміщення графіки і контенту на Інтернет сторінках	0,2	
Всього		0,8	

Складемо штатний розклад виконавців:

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (ІТ-менеджер)	1	16779	33558
Продакт-менеджер	0,4	15000	12000
Інженер-програміст	2,6	16000	83200
Інженер-електронщик	0,28	14000	7840
Інженер-системотехнік	0,1	15000	3000
Адміністратор мережі	0,1	15000	3000
Системний програміст	0,1	15000	3000
Дизайнер WEB	0,1	15000	3000
Інженер-верстальник	0,1	15000	3000
Бухгалтер-економіст	0,1	15000	3000
Всього за період розробки	$R_{сп}$=4,88	-	$Ф_{роб}$=154598

Розрахуємо середньоденну зарплату одного виконавця:

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

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

$$З_{сд} = \frac{154598}{4,88 \cdot 48} = 660 \; грн.$$

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 ТОВ науково-дослідницького консалтингового підприємства «Пекторaль» (м. Кіровоград, вул. Глинки 16) ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 400…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 37 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:

$$Б_{уд} = 6 \cdot 8 \cdot 20000 = 960000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 96000 грн.

Балансова вартість інвентарю розраховуємо за нормою 15000 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.

$$I_{нв} = 6 \cdot 15000 = 90000 \text{ грн}$$

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

Специфікація на обчислювальну техніку наведена в таблиці 7.7. Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу фірми Brain за 30.10.23 – джерело http://brain.com.ua

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		12721
Системний блок		7721
Процесор	AMD Ryzen 5 4500 AM4, 6 ядер, 12 потоків, 3.6 GHz, 4.1 GHz, TDP – 65 Вт, 7nm	
Системна плата	GIGABYTE A520M H, AM4, DDR4, 64 ГБ, 5100 MHz, LAN – 1 Гбіт/с, DVI, HDMI, 1 x M.2 2280, 4 x Sata 6.0 Gb/s, Micro-ATX	
Відеокарта	Radeon R7 350 2Gb Afox (AFR7350-2048D5H4-V3) PCI-Express 3.0, 2 ГБ, GDDR5, 128 Bit	
Жорсткий диск	SSD M.2 2280 240GB Apacer (AP240GAST280-1) 240 GB, TLC, M.2, SATA III (6Gb/s)	
Оперативна пам’ять	DDR4 8GB 2666 MHz Kingston (KVR26N19S8/8) DDR4, 8 ГБ, В наборі – 1	
DVD-привод	DVD±RW ASUS DRW-24B5ST Black Bulk	
Корпус	Vinga Apache-500W, Miditower, ATX, Micro – ATX, Mini – ITX, PSU – 500 Вт	
Кардрідер внутрішній	Transcend TS-RDF8K USB 3.0	
Система охолодження	Vinga Q6 – INTEL, AMD, сокет – FM2+, 1151, AM4, 1150, 1155, AM2, AM2+, AM3, FM1, FM2, 1700, AM5, 1152, 775, 1156	
інше	Клавіатура, мишка	

Продовження таблиці 7.5

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Монітор	Монітор BenQ GL2450HM Black	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37	2800
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійного живлення	Powercom BNT-600AP USB	1400

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

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – 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	8	12721	10176,8	111944,8
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2800	280	3080
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	133576,3

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	960000	-	-
2. Передавальні пристрої	96000	-	-
Всього по групі	1056000	5	52800
Група 4			
3. Обчислювальна техніка	137536	-	-
Всього по групі	137536	30	41260,8
Нематеріальні активи			
4. Нематеріальні активи	13000	25	3250
Група 5, 6			
5. Вимірювальні пристрої	9031	25	2257,75
6. Транспортні засоби	121875	20	24375
7. Господарський інвентар	90000	25	22500
Всього по групі	220906	-	49132,75
Разом	$К_р$ = 1427442		$А_р$ = 146443,55

Примітка: вартість автомобіля ГАЗ Газель взята по даним електронного ресурсу http://www.avtopoisk.ua, що складає 121875 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.

$$З_о = 660 \cdot 112/130 = 569 \text{ грн}$$

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, %

$$З_д = 569 \cdot 10 \cdot 0{,}01 = 57 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=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, %

$$С_{оц} = 0{,}01 \cdot 22(569+57) = 138 \text{ грн}$$

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

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

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 569 \cdot 15 \cdot 0{,}01 = 85 \text{ грн}$$

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, картриджів, тонеру, грн., $N_е$ – кількість екземплярів програм, шт.

Згідно виданих норм приймаємо 1/3 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=210 грн., визначаємо вартість паперу за період розробки $N_м$=2 міс:

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

$$З_{М1} = 210 \cdot (1/3) = 70 \text{ грн.}$$

Згідно виданих норм до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків в кількості 30 примірників:

$$З_{М2} = \sum Ц_д., \quad (7.17)$$

де $Ц_д$ – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 30,7 грн/шт., DVD-R LG 4,7Gb, 16x speed Cake box – 30,7 грн/шт.

$$З_{М2} = 30 \cdot 30,7 = 922 \text{ грн.}$$

Згідно виданих 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.

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

$$З_М = (70 + 922 + 1702)/130 = 21 \text{ грн.}$$

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, %

$$О_п = 569 \cdot 15 \cdot 0,01 = 85 \text{ грн}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 130 прим.)

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

де Ар – загальна річна сума амортизаційних відрахувань, грн.

$$A_м = 146444 \cdot 2/(130 \cdot 12) = 188 \text{ грн}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

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

$$C_п = 569+57+138+85+21+85+188 = 1143 \text{ грн.}$$

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 $P_с$ – рівень рентабельності, %

$$П_р = 0{,}01 \cdot 50 \cdot 1143 = 571{,}5 \text{ грн.}$$

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	$З_о$	569
2. Додаткова зарплата виконавців	$З_д$	57
3. Відрахування на соціальні потреби	$С_{оц}$	138
4. Загальногосподарські витрати	$Г_{осп}$	85
5. Витрати на матеріали	$З_М$	21
6. Освоєння нових операційних систем, мов програмування	$О_п$	85

Продовження таблиці 7.9

1	*2*	3
7. Амортизація основних фондів	$А_м$	188
8. Повна собівартість програмного забезпечення	$С_п$	1143
9. Плановий прибуток	$П_р$	571,5
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	1714,5
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	*ПДВ*	342,9
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	*Ц*	2057,4

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	–	2057
Всього капітальних витрат	–	2057

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 системи	$З_р$	10065	2684
2. Витрати на амортизацію	$З_{ам}$	0	514
Всього витрат за рік	$І$	10065	3198

Витрати на технічне обслуговування і підтримку працездатності системи:

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

де $Т_р$ – кількість годин обслуговування за рік, *год.*,

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

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 75 годин на рік до 20 годин на рік, тому витрати на технічне обслуговування зменшилися з

$$З_{р\,баз} = 75 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 10065 \text{ грн.}$$

до

$$З_{р\,нов} = 20 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 2684 \text{ грн.}$$

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	–	2057	–	514,25
Всього відрахувань	-	–	2057	–	514,25

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

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.24)$$

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

$$Ев = (1714,5 - 1143) \cdot 130 - (0,05 \cdot 1056000 + 0,3 \cdot 137536 +$$
$$+0,2 \cdot 121875 + 0,25 \cdot 99031 + 0,25 \cdot 13000) \cdot 2/12 = 49888 \; грн.$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_p^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.25)$$

де: K_p^* – балансова вартість основних фондів розробника.

$$T_в = \frac{1427442}{(1714,5 - 1143) \cdot 130 \cdot 12 / 2} = 3,2 \; років$$

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.	130
2. Повна собівартість розробленої програми	Грн.	1143
3. Ціна розробленої програми	Грн.	1714,5
4. Плановий прибуток від реалізації розробленої програми	Грн.	571,5
5. Рентабельність програмної продукції	%	50
6. Об’єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1427442
7. Загальний прибуток від реалізації програмної продукції	Грн.	74295
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	49888
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	3,2
10.Об’єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	2057
11.Величина економічного ефекту у користувача програмної продукції	Грн.	6353
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Роки	0,3

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \qquad (7.26)$$

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $К_б$, $К_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (10065\text{-}3198)\text{-}0{,}25 \cdot 2057 = 6353 \text{ грн.}$$

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

$$T_{сп} = \frac{К_н - К_б}{I_б - I_н} \tag{7.27}$$

$$T_{сп} = \frac{2057}{10065 - 3198} = 0{,}3 \ роки$$

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

Загальна комп'ютеризація суспільства призвела до того, що використання комп'ютерів стало повсюдним у всіх сферах економіки та народного господарства. Застосування персональних комп'ютерів і ЕОМ дозволило значно підвищити продуктивність праці, змінити характер і зміст праці. Комп'ютеризація, поряд з незаперечними перевагами, тягне за собою і багато проблем. Для того, щоб активне застосування комп'ютерних технологій не стало додатковим чинником погіршення здоров'я, вкрай необхідно щоб робоче місце відповідало гігієнічним вимогам. Темою дипломного проекту є розробка та дослідження та реалізація програмного продукту, тому актуально буде розгляд умов праці програміста.

В охорону праці включають санітарно-гігієнічні, лікувально-профілактичні та організаційно-технічні системи правових і соціально-економічних заходів.

В кожній ІТ компанії є трудові відносини з працівниками. Згідно закону України "Про охорону праці" [1] кожна компанія впроваджує заходи з охорони праці. Реалізується трудові відносини з вживанням необхідних засобів з охорони праці та розробки відповідних документів:

– Інструкцій з охорони праці по кожній професії і загальні;

– Положення про охорону праці;

– Накази з охорони праці;

– Журнали реєстрації та інструктажу.

Роботодавець створює відділ який працює відповідно до типового положення, яку затверджується центральним органом виконавчої влади і забезпечує виконання вимог державної політики у сфері охорони праці.

За недотриманням вимог, керівники ІТ компаній можуть бути притягнуті до відповідальності, яка виглядає у виді накладання штрафу. Якщо в результаті

порушення умов охорони праці є постраждалі працівники то керівні особи ІТ компаній притягуються до кримінальної відповідальності.

Законом України “Про охорону праці” [1] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [5], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

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

Ці шкідливі фактори можуть привести до професійних захворювань.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98, та «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

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

8.2 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

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

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	3,5
Довжина	3,5
Висота	3

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

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	6,1
Обсяг, V	$м^3$	не менше 20.0	18,4

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

У зазначеному приміщенні працює 2 людей. За даними, які наведено у табл. 8.1 та табл. 8.2, можна зробити висновок, що площа приміщення у розрахунку на одно робоче місце програміста відповідають нормативним вимогам (Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про

затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», а об'єм – ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2] та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»).

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови Головного державного санітарного лікаря України [5], робота, яка виконується в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря у приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

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

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

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура, °С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-23	45-60	0,11
Тепла	23-25	50-70	0,1	23-25	52-70	0,1

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

В літню пору застосовується кондиціонер.

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

У приміщенні знаходяться наступні джерела шуму: принтер HP Laser 107a, електродвигуни вентиляторів ЕОМ.

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

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5 – 28 – 2006 р можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному

висвітленні повинна становити 300 лк. Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.3 Розробка заходів з умов поліпшення охорони праці

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

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

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значеннь;

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

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

8.4 Розрахункова частина

Завдання: розрахувати *штучне освітлення робочого приміщення.*

Початкові дані: ширина *робочого* приміщення: 3,5 м.; довжина –3,5 м.; висота – 3 м.

Розрахунок штучного освітлення проводемо за методом коефіцієнта використання світлового потоку.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою *[9]*:

$$F=ESKZ/n,$$

де: *F* – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; *E = 300 Лк;*

S – площа освітлюваного приміщення (у нашому випадку *S*=3,5 х 3,5 = 11,9 $м^2$);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку *Z = 1,1*);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K = 1,5);*

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін **($\rho_{стін.}$)** і стелі **($\rho_{стелі}$),** значення коефіцієнтів дорівнюють **$\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%** *[6]***.**

Обчислимо індекс приміщення за формулою:

$$i=S/(h(A+B)),$$

де: S – площа приміщення, $S = 11{,}9$ *м²;*

h – розрахункова висота підвісу, $h = 3$ *м;*

A – ширина приміщення, $A = 3{,}5$ *м;*

B – довжина приміщення, $B = 3{,}5$ *м.*

Підставимо всі значення у формулу та визначемо індекса приміщення: *i=0,57.*

Знаючи індекс приміщення, за знаходимо *n = 0.29 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників відповідного типу [6]).* Підставимо всі значення у формулу, визначемо світловий потік: *F=13182 Лм.*

Для штучного освітлення приміщення використовуються *LED панель MAXUS ASSISTANCE PRO 80W 5000K WHITE (M1052480531)*, світловий потік яких $F_л = 8000$ Лм.

Число світильників визначається по формулі **[5]**:

$$N=F/F_л$$

де: F – світловий потік,

$F_л$ – світловий потік одного *світильника*.

$$N= 13182/ 8000=1{,}6 \text{ шт.}$$

Приймаємо необхідну кількість світильників 2 шт.

8.5 Висновки до розділу

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

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

Список використаних джерел інформації

1. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

2. 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

3. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

4. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

5. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. – Кіровоград: КІСМ, 1997. – 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358 (дата звернення: 16.06.2023).

6. 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

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

8. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький: ЦНТУ, 2022. – 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення: 16.06.2023).

9. Методичні рекомендації до виконання розділу «Заходи з охорони праці та техніки безпеки» у магістерській дисертації / Л.Д. Третякова; М-во освіти і науки України, Національний технічний університет україни «Київський політехнічний інститут» – Київ, КПІ, 2014. – 26 с. – Режим доступу до ресурсу: http://surl.li/dhulo (дата звернення: 16.06.2023).

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи управління файлами на iPhone.

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 управління файлами на iPhone.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем управління файлами на iPhone.

– Досліджена система управління файлами на iPhone.

– На основі отриманих результатів досліджень створена програмна реалізація системи управління файлами на iPhone.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання управління файлами на iPhone.

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 iOS.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм UMAC.

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 6353 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,3 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Коваленко В.О. Дослідження та програмна реалізація системи управління файлами на iPhone // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2023.

2. Adam Freeman. Pro Go The Complete Guide to Programming Reliable and Efficient Software Using Golang. Apress Media. 2022. 1078 p.

3. Fernando Doglio. Skills of a Successful Software Engineer. Manning. 2022. 182 c.

4. M. Holmes He. Creating Apps with React Native. Apress Media. 2022. 445 p.

5. Maurício Aniche. Effective Software Testing. Manning Publications. 2021. 372 p

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

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

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

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

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

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

12. 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.

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

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

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

16. 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.

17. 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.

18. 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.

19. 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.

20. 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

21. 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.

22. 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.

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

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. 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.

34. Вінтенко Б.Ю., Смірнов О.А., Коваленко А.С., Смірнов С.А., Буравченко К.О. «Дослідження вимог міжнародних стандартів IEC60880 та IEC62138 з розробки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 3(73), С. 155-166.

35. 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.

36. 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.

37. 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.

38. 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.

39. 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.

40. 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.

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

42. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan

D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

43. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

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

45. O. Smirnov, O. Kovalenko, A. Kovalenko, S. Smirnov, V. Vialkova. The mathematical model of the testing technology for DOM XSS vulnerabilities. Scientific & practical cyber security journal (SPCSJ) Vol 2 Issue 1, 22-28 pp. [Электронный Журнал]. Georgia. Tbilisi: SCSA – 2018.

46. Oleksii Smirnov, Oleksandr Kovalenko, Jamil Al-Azzeh, Anna Kovalenko, Serhii Smirnov. Qualitative risk analysis of software development. Asian Journal of Information Technology. – Volume 17(3). – Medwell Journals. – 2018. – P. 218-230.

47. Смірнов О.А., Коваленко О.В., Коваленко А.С., Смірнов С.А. Розробка методу передтестової компіляції й розподілу доступу. Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницький. 19-20 квітня 2018р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215

48. Smirnov Oleksii, Kovalenko Oleksandr, Kovalenko Anna, Smirnov Serhii. Method of testing the DOM XSS vulnerability. International Conference «Information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces “Alexandru cel Bun”. 2017. P7.

49. 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.

50. 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.

51. 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.

52. 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.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРМ-122.23.0010.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		Коваленко В.О.			Дослідження та програмна реалізація системи управління файлами на iPhone	Літ.	Аркуш 1 / Аркушів 6
Перевірив		Доренський О.П.					
Н. Контр.		Коваленко А.С.				ЦНТУ КН-22М-1	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи управління файлами на iPhone.

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. № 32-13 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 управління файлами на iPhone.

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 управління файлами на iPhone;

– 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 обільному пристрої під керуванням iOS 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 iOS.

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 повинен бути розглянутий аналіз санітарно-гігієнічних умов праці на робочому місці програміста.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 107 аркушів.

10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

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

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 22.12.2023 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

_____ Доренський О.П.

Дослідження та програмна реалізація системи управління файлами на iPhone

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 44

Літера: РП

Кропивницький – 2023 року

Folder.m – робота з папками

```
      /*

     Folder.m
     iPhoneConnection

      Copyright (c) 2023 Коваленко Владислав Олегович

      */

#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) 2023 Коваленко Владислав Олегович

 */

#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) 2023 Коваленко Владислав Олегович

    */

#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) 2023 Коваленко Владислав Олегович


       */

#import <Cocoa/Cocoa.h>


@interface File : NSObject {

    NSString *location;

}

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


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

@end
```

AFCInterface.m – Робота з сенсорним екраном

```
     /*

 AFCInterface.m
 iPhoneConnection

  Copyright (c) 2023 Коваленко Владислав Олегович

  */

// 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) 2023 Коваленко Владислав Олегович

      */

#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) 2023 Коваленко Владислав Олегович
       */


#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) 2023 Коваленко Владислав Олегович
       */

#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) 2023 Коваленко Владислав Олегович
*/

#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) 2023 Коваленко Владислав Олегович

       */

#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) 2023 Коваленко Владислав Олегович

 */

#import <Cocoa/Cocoa.h>


@interface DisconnectedView : NSView {

}

@end
```

AFCFactory.m – підключення / відключення повідомлень

```
	/*

	AFCFactory.m
	iPhoneConnection

	 Copyright (c) 2023 Коваленко Владислав Олегович
	 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) 2023 Коваленко Владислав Олегович
       */

// 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) 2023 Коваленко Владислав Олегович

	 */

#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
```

AFCDevice.m – клас-оболонка для класу AFCInterface

```
	/*

	AFCDevice.m
	iPhoneConnection

	 Copyright (c) 2023 Коваленко Владислав Олегович

	 */

// 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) 2023 Коваленко Владислав Олегович

	 */

#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
```