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 диспетчеризації файлів у мобільній операційній системі iOS 16”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-22М-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Пархоменко Д.О.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Дрєєв О.М.
«____» _____ 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 123 *"Комп'ютерна інженерія"*

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

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

Пархоменку Дмитру Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16*

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

затверджені наказом вищого навчального закладу № 35-13 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 диспетчеризації файлів у мобільній операційній системі iOS 16*

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

Пархоменко Д.О. Дослідження та програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи диспетчеризації файлів у мобільній операційній системі iOS 16.

Метою розробки є дослідження та програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16.

Об'єктом дослідження є процес диспетчеризації файлів у мобільній операційній системі iOS 16.

Предметом дослідження є методи диспетчеризації файлів у мобільній операційній системі iOS 16.

Методи дослідження базуються на методах комп'ютерної інженерії, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16.

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 ОС iOS 16.

Програму розроблено в середовищі Objective-C.

Ключові слова: комп'ютерна інженерія, диспетчеризація файлів

ABSTRACT

Parkhomenko D.O. Research and software implementation of the file dispatching system in the mobile operating system iOS 16. 123 Computer engineering. 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 dispatching system in the iOS 16 mobile operating system.

The purpose of the development is the research and software implementation of the file dispatching system in the iOS 16 mobile operating system.

The object of the study is the process of dispatching files in the iOS 16 mobile operating system.

The subject of the study is file dispatching methods in the iOS 16 mobile operating system.

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

The result of the work is the software implementation of the file dispatching system in the iOS 16 mobile operating system.

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

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

The program can be used on a mobile device running iOS 16.

The program was developed in the Objective-C environment.

Keywords: computer engineering, file dispatching

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. Відсутність, як такої, файлової системи для кінцевого користувача пристроїв на iOS і, як наслідок, розподіленість файлів і документів різного формату по окремих додатках накладають певні труднощі й незручності для продуктивного виконання завдань у будь-якому місці. Це все послужило появі в AppStore великої кількості додатків які дозволяють у якійсь мері вирішити дані проблеми. Згодом вони еволюціонували, здобували нові функції, ставали зручніше й могли читати усе більше й більше різних форматів. Крім функції організації зручної файлової ієрархії, з якою, до речі, вони справляються, ці програми дозволяють нам відкривати для читання файли з такої створеної структури. Це можуть бути як, у першу чергу, офісні документи, текстові файли або документи PDF, так і архіви, зображення, відео й аудіо записи. Відразу варто відзначити, що мова йтиме не про пакети для створення й редагування файлів Word, Excel і PowerPoint, які вимагають окремої розмови й перебувають зовсім в іншому ціновому діапазоні, а про, так сказати, «всеїдні» читалки.

iOS 16 – операційна система, розроблена корпорацією Apple.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16.

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

– Огляд існуючих систем диспетчеризації файлів у мобільній операційній системі iOS 16.

– Дослідження системи диспетчеризації файлів у мобільній операційній системі iOS 16.

– Програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16.

Об'єктом дослідження є процес диспетчеризації файлів у мобільній операційній системі iOS 16.

Предметом дослідження є методи диспетчеризації файлів у мобільній операційній системі iOS 16.

Методи дослідження базуються на методах комп'ютерної інженерії, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод диспетчеризації файлів у мобільній операційній системі iOS 16.

– Розроблено вітчизняний продукт диспетчеризації файлів у мобільній операційній системі iOS 16, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі диспетчеризації файлів у мобільній операційній системі iOS 16.

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 диспетчеризації файлів у мобільній операційній системі iOS 16, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для диспетчеризації файлів у мобільній операційній системі iOS 16 та пропонує найширший вибір операцій по роботі з файлами: перегляд, створення, виправлення, архівацію, видалення, переміщення, відправлення по e-mail і AirPlay, роботу методом драг-н-дроп і багато чого іншого.

Система диспетчеризації файлів у мобільній операційній системі iOS 16 підтримує широкий перелік типів файлів, починаючи від мультимедійних, закінчуючи базами даних і скриптами. Програма відтворює відео й музику так само, як нативний плеєр і забезпечує доступ до файлів на iOS-пристрої з комп'ютера за допомогою локальних веб-адрес і протоколу Bonjour (Air Browser).

Система диспетчеризації файлів у мобільній операційній системі iOS 16 поширюється по умовно безкоштовній моделі.

Можливості системи диспетчеризації файлів у мобільній операційній системі iOS 16:

– Перегляд файлів: відтворення мультимедійних файлів, HEX-редактор, текстовий редактор, потужний редактор SQLite, термінал DEB.

– Можливість виконувати скрипти й додатка парою кліків.

– Можливість відправляти файли на комп'ютер і iOS-пристрою за допомогою Air Browser.

– Можливість шукати файли й папки з використанням RegEx.

– Можливість переглядати файли списком, сіткою й у режимі передперегляду.

– Можливість використовувати команди вирізати, копіювати, вставити.

– Убудований архіватор для файлів zip, tar і gz.

1.2 Область застосування

Областю застосування системи диспетчеризації файлів є мобільна операційна система iOS 16.

Нові можливості iOS 16:

– Фотографії – пошук по місцю й часу, а також нові можливості редагування фотографій [3].

– Повідомлення – можливість відправляти аудіозаписи й карту з місцезнаходженням у діалог. Тепер можна швидше переслати тільки що зняті відеозаписи й фото, а також установлювати функцію «Не турбувати» на потрібні діалоги [4].

– Quicktype – угадує можливі за змістом слова, засновані на розмові, під час друкування пропозицій [5].

– Family Sharing – можна позначати до шести контактів як членів родини й швидко ділитися з ними фотографіями, покупками додатків і музики, місцем розташування й інше [6].

– iCloud Drive – можливість зберігати в хмарі будь-які види файлів з наступним їхнім редагуванням на різних пристроях [7].

– HealthKit – організація відомостей про здоров'я в одному додатку [8].

– Spotlight – пошук став більше глобальним і тепер можна шукати різну інформацію й за межами телефону/планшета [9].

– Віджети – тепер у Центр Повідомлень можна встановлювати віджети від сторонніх розроблювачів.

– Сторонні клавіатури – уперше компанія Apple дозволила стороннім розроблювачам створювати альтернативні клавіатури, які зможуть замінити стандартну клавіатуру.

– Можливість для убудованих і зовнішніх додатків обмінюватися даними, передавати один одному керування для обробки цих даних.

– Handoff (Continuity) – при використанні OS X Yosemite можливий легкий перенос активностей між комп'ютером, iPhone і iPad, наприклад можна почати писати лист на одному пристрої, а продовжити на іншому.

– Використання випадкових MAC-адрес при пошуку Wi-Fi мереж у сплячому режимі (фактично працює лише на телефонах, з відключеним стільниковим зв'язком і Location Services, а також без активного Wi-Fi підключення) [10]

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Проведемо огляд найбільш популярних і дійсно зручних файлових менеджерів – Documents By Readdle і Phone Drive. Що стосується ціни, більшість із цих додатків безкоштовні або коштують щодо невеликих грошей. Мені, як студентові, що закінчує університет, щоб усе встигати, просто життєво необхідно мати доступ і засоби маніпулювання своїми документами в будь-якому місці й у будь-який час. Тому такими читалками я почав користуватися досить давно й перепробував досить велику їхню кількість. Більшість із них нефункціональні, тому що вони або моторошно незручні, або постійно вилітають, або мають убогого функціонала й працюють тільки з декількома типами файлів. Але є й гідні додатки. Поєднує їх те, що вони досить непогано справляються із зазначеними вище функціями й мають логічний і зрозумілий інтерфейс. Також для них характерний загальний недолік – труднощі читання документів Word (на жаль, на сьогодні ця проблема так і не вирішена). І в першу чергу, це проблеми зі шрифтами (деякі не розпізнаються) і таблицями (відбувається іноді зсув тексту в них), а також взагалі не читаються схеми, побудовані засобами малювання Word, хоча ті ж убудовані в документ схеми Visio, графіки MathCad або рисунки відображаються коректно. Саме тому я волію зберігати на своєму планшеті документи у форматі PDF (наприклад, конспекти лекцій). Розходжень же в Documents і Phone Drive небагато, але все-таки вони є.

Documents by Readdle

Цей файловий менеджер з'явився в магазині зовсім недавно і є безкоштовним. Він сподобався багатьом користувачам у першу чергу за рахунок

простоти й адекватності в роботі з більшою кількістю різних форматів. А від інших додатків він виділяється можливістю читання книг і вихідних кодів програм з підсвічуванням синтаксису, а також багатими можливостями для синхронізації.

Рисунок 2.1 – Інтерфейс користувача Documents by Readdle

Зовні вид додатка не викликає в мене дорікань, всі як і мабуть, на мій погляд. Дизайн виконаний у світле сірих тонах. У лівій частині панель для швидкого доступу до основних директорій – локальній папці з нашими документами, загальним файлам iCloud, хмарним сховищам, недавно відкритим файлам і до браузера.

Основну ж частину екрана займає область, у якій відповідно відображаються наші файли й папки. Зверху можна задати тип сортування й вид відображення вмісту. Труднощів в освоєнні ніщо не викликає.

Для виконання всіх дій з нашим контентом призначена кнопка Edit, розташована на верхній панелі. Вибравши необхідні документи або папки, ми можемо їх копіювати й переносити в інші папки, перейменовувати, видаляти, додавати в архів і відправляти поштою або завантажити, скажемо, в SkyDrive. А крім додавання нових папок ще можна створювати текстові документи.

Також зверху є кнопка пошуку. Шукати ми можемо як по іменах файлів, так і по вмісту документів.

Загалом, у плані зручності й логічності роботи все в нормі.

Що стосується підтримуваних форматів, Documents дозволяє відкривати тестові файли, документи Microsoft Office (Word, Excel і PowerPoint) і iWork, PDF, аудіо й відео файли популярних форматів (mp3, aac, wav, mp4, mov та інші), зображення (jpg, gif, bmp, png, tiff), а також розпаковувати архіви популярних форматів. Крім цього, додаток дозволяє читати книги у форматах epub і fb2 і відкривати файли з вихідним кодом, підсвічуючи синтаксис відповідної мови програмування (C, C#, VC і Lisp він знає, а от, наприклад, популярний у минулому файл, написаний на Delphi, відкрився як звичайний текстовий документ).

Такої можливості конкуренти не надають, а це може придатися людям, що займаються розробкою ПЗ.

Робота з архівами (Phone Drive також підтримує таку можливість) є для мене важливою перевагою, тому що убудованих для цього в iOS засобів немає й окремих додатків не так багато.

Читання офісних документів і PDF цілком комфортне. Сподобалася можливість перемикання для таких документів зі звичайного режиму в нічний. А для PDF – виділення тексту макреком, підкресленням або закреслюванням.

Плеєр музики виглядає цілком пристойно. Основні елементи керування на своєму місці й виконані в дусі Apple. Та й особливих вимог у таких додатках до нього не пред'являється. Теж стосується й відео, плеєр нічим не відрізняється від убудованого в iOS.

Перегляд фотографій і зображень також звичний. Зміна зображень здійснюється свайпом, а для перегляду доступні всі файли з папки, у якій ви відкрили дане фото.

Крім синхронізації по iCloud, Documents надає найбільший список інших хмарних сервісів. Це можуть бути як Dropbox, SkyDrive, Google Drive, Box, CloudMe і інші сховища.

Ще однією перевагою я б хотів відзначити можливість гнучкого налаштування додатка. Це дозволяє нам щонайкраще адаптувати додаток під наші вимоги. Тут можна встановити пароль (Phone Drive також підтримує дану важливу опцію), включити індексування або настроїти окремі оглядачі.

Правда є й нелогічність у деяких опціях. Наприклад, щоб перевести оглядач вихідного коду в режим редагування, необхідно зайти в меню налаштувань і включити таку функцію, а щоб знову включити підсвічування синтаксису й перейти до читання – необхідно зробити зворотне. Чому б не зробити перемикання безпосередньо, коли файл відкритий? Теж і стосується горизонтального й вертикального прокручування.

Іншим недоліком є відсутність української мови, хоча це некритично, у додатку й так все зрозуміло.

Phone Drive

Інший файловий менеджер володіє меншим, з погляду кількості підтримуваних форматів, функціоналом, але відрізняється більше гнучкою роботою з медіа й широкими можливостями для передачі файлів. Він з'явився в AppStore досить давно й згодом його функціонал поліпшувався (скажемо, підтримка Dropbox і iCloud додана недавно). Коштує Phone Drive 50 гривень. Є й безкоштовна версія, але її можна порекомендувати тільки для ознайомлення.

Як і Documents by Readdle, додаток має зручний лаконічний дизайн, тільки вже більше темний. З лівої сторони відображена ієрархія наших папок, а основна частина відведена під їхній уміст (варто помітити, що обоє учасника огляду мають приблизно однакову структуру, характерну для цього класу додатків). А от основні елементи керування розташувалися внизу екрана.

Рисунок 2.2 – Інтерфейс користувача Phone Drive

Доступ до браузера, налаштуванням і пошуку винесений на першу кнопку. Вибір зовнішнього вигляду вмісту й сортування рознесені на різні кнопки зі зрозумілими піктограмами. Хоч таке рішення й вимагає зайвих дій, у порівнянні з Documents, але ніякої незручності це не викликає.

Нажавши кнопку змінити, ми також можемо виконувати різні дії над нашими файлами й папками – список їх такої ж, як і в попереднього додатка.

Крім створення папок і текстових документів, Phone Drive дозволяє нам створювати плейлисти, зробити швидкий знімок або записати відео, не виходячи з додатка, а також записати звук. І це саме мене залучило в першу чергу в цьому додатку. Це дуже зручно, коли швидко можна зробити фотографію й прикріпити її до інших файлів в архіві, а потім відправити поштою. А плейлисти схвалять ті люди, які через убудований браузер завантажують музику на свій пристрій, обходячи при цьому iTunes.

Але для багатьох підтримка файлів мультимедіа взагалі не так важлива, і вони можуть порахувати все це зайвим, але не варто забувати, документи для роботи можуть бути не тільки текстовими, наприклад диктофонні записи, що навчають відеоролики, або знімки сканера. І я вважаю, не має змісту зберігати такі файли разом з музикою, що ми слухаємо, фільмами й фотографіями. Та й підтримка великої кількості форматів робить такі додатки більше універсальними.

Phone Drive може працювати з тими ж типами файлів, що й Documents, за винятком книг і вихідних кодів (вони відкриваються як текст). Але відсутність «книжкових» можливостей, а тим більше підсвічування синтаксису у файлах з розширенням, знайомим далеко не кожному, критичним назвати не можна. Хоча це, безсумнівно, плюс для Documents by Readdle. Розходжень у читанні офісних файлів ніяких ні, та й PDF-Документи переглядати однаково зручно. Є підтримка закладок, правда нічного й денного режимів немає.

Відеоплеєр, як і в Documents, у точності повторює стандартний. Так це й до кращого – нічого зайвого й всі там, де повинне бути. А от музичний плеєр виглядає небагато інакше, хоча функціональність та ж.

Перегляд фотографій мені менш сподобався за рахунок зайвих кнопок, але, проте, всі теж зручно, а з додаткових опцій відправлення на печатку по AirPrint і слайд шоу.

Якщо порівнювати двох учасників огляду по швидкості відкриття файлів і плавності роботи з текстовими документами, то отут розходжень немає. Обоє вони на висоті й навіть багатосторінкові документи Word або PDF завантажуються досить швидко.

Інша важлива перевага над Documents – це можливість передачі файлів будь-якого типу з комп'ютера на пристрій по Wi-Fi прямо через провідник або браузер.

Досить просто ввести ftp або http адреса пристрою, і ми можемо переносити документи як на звичайну флешку.

Крім цього, Phone Drive дозволяє обмінюватися файлами з іншими пристроями на iOS, на яких він теж установлений, через Bluetooth і Wi-Fi.

Такі особливості сподобається великій кількості користувачів.

З недоліків я б відзначив досить посередні можливості для синхронізації, у порівнянні з Documents. Це загальні папки iCloud і Dropbox. Також додаток може віджахнути багатьох людей своєю ціною, при рахунку, що є безкоштовний аналог зі схожим функціоналом.

Підбиваючи підсумок, можна сказати, що Documents by Readdle і Phone Drive гідно справляються з основними завданнями на тлі конкурентів і мають свої відмінні риси. Що стосується мене, то я почав користуватися Phone Drive ще до появи Documents, і він мене цілком улаштовує, але остаточний вибір убік того або іншого файлового менеджера поки зробити не готов, обоє вони гарні.

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 диспетчеризації файлів у мобільній

операційній системі iOS 16.

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

Оновлення програми Files в iOS 16 і iPadOS 16 наближають її до macOS Finder. Ось що нового.

Програма Files надає базові функції керування файлами на iPhone та iPad. Хоча він не такий надійний, як Finder на Mac, він може виконувати більшість того, що потрібно користувачеві для керування файлами.

Файли починалися як спосіб перегляду всього, що зберігається в iCloud Drive, потім розширилися до сторонніх хмарних служб, як-от Dropbox, і, нарешті, завершилися параметрами локального та мережевого зберігання. Тепер набір функцій знову розширено кількома вкрай необхідними оновленнями для навігації та керування файлами в iOS 16 і iPadOS 16.

Оновлення програми Files

Більшість змін, внесених у програму Files, спрямовані на те, щоб зробити парадигми взаємодії подібними до macOS для кращої паритетності екосистеми. Ці нові оновлення спрощують пошук, маніпулювання та впорядкування файлів.

Функції збереження, відкриття та передачі

Функцію «Зберегти у файли» було вдосконалено за допомогою нового модального вікна. Кожного разу, коли користувач зберігає щось у програмі «Файли», з’являється весь інтерфейс програми з параметрами вибору місця збереження, додавання тегу та перейменування файлу.

Збереження, переміщення та відкриття файлів мають кращі засоби керування користувачами

Відкриття файлів за допомогою програм сторонніх розробників отримує подібний інтерфейс користувача, якщо розробник правильно націлює програму Файли як місце зберігання. У модальному вікні показано всі інтерфейси навігації

для пошуку файлу, включаючи нові параметри навігації, описані нижче.

Під час переміщення файлу в інше місце зберігання, наприклад, з однієї хмарної служби в іншу або на зовнішній SSD, з'являється індикатор передачі файлу.

Переглянути параметри та навігацію

Apple покращила режим перегляду списку за допомогою кращих засобів керування організацією. Файл або папка відображаються ліворуч, як зазвичай, із датою зміни та розміром файлу праворуч із заголовками стовпців синім кольором.

Навігація папками та керування файлами стає легшою завдяки додатковим елементам керування

Торкніться заголовка стовпця, щоб упорядкувати список за цією точкою даних, і торкніться його ще раз, щоб змінити порядок. Групування за типом файлу, датою, розміром або тим, хто ним поділився, пропонує більше варіантів організації.

Панель інструментів навігації також отримала невелику, але бажану зміну. Замість того, щоб відображати батьківську папку або попередню папку у верхньому лівому куті, програма «Файли» тепер показує стрілки вперед і назад біля назви поточної папки.

Торкання назви папки відкриває навігаційне подання останніх папок і пропонує кілька параметрів керування, як-от перейменування, копіювання та переміщення.

Упорядковуйте файли та папки за допомогою груп і параметрів списку

Якщо в меню перегляду вибрати «Параметри перегляду», відобразяться параметри «Групувати за» для будь-якого заданого режиму перегляду, а також кнопка «Показати всі розширення», про яку ми розповімо нижче.

Нові дії меню

Під час взаємодії з файлом доступна кілька нових взаємодій залежно від типу файлу та контексту. Вони приховані під меню «клацніть правою кнопкою

миші» або натисніть і утримуйте меню.

Використання «Отримати інформацію» для папок тепер покаже розмір вмісту папки. Раніше тут не відображалися дані про розмір. Крім того, розділ «де» показує файлову структуру, що веде до вибраного файлу або папки.

У меню програми «Файли» є нові параметри для зміни файлів

При виборі певних типів файлів у меню з'являться нові швидкі дії. Наприклад, швидкі дії із зображенням включають обертання зображення, перетворення на PDF або видалення фону.

Якщо вибрано «Видалити фон», нове зображення PNG зберігається з об'єктом у тій же папці.

Макет «Стовпці» пропонує гібридне подання інформації та кнопок дій, які зазвичай є, коли зображення чи файл вибрано в меню вибору файлу. Кнопка гамбургера показує додаткові дії з файлами, подібні до перерахованих раніше.

Крім того, деякі дії можна виконувати з кількома вибраними файлами. Наприклад, виберіть кілька файлів зображень, щоб усі вони були додані до файлу PDF, який зберігається окремо.

Інші пакетні операції включають зміну розширень файлів, видалення фону з усіх вибраних зображень або створення нової папки з вибраними елементами. Проте програма Файли не може пакетно перейменовувати файли.

Розширення файлів

Розширення файлів повідомляють вам, з яким файлом ви маєте справу, наприклад, файл зображення може бути JPEG, PNG або HEIF. Тепер користувачі можуть вільно конвертувати зображення між цими типами розширень за допомогою дії «Перетворити зображення» в меню швидких дій.

Змінюйте розширення файлів у програмі iOS 16 Files

Для більш досвідчених користувачів скористайтеся перемикачем «Показати всі розширення» в меню макета, щоб побачити розширення файлу після імені файлу. Якщо це ввімкнено, просто перейменуйте файл і змініть розширення, яке відображається після крапки, щоб фактично змінити тип файлу.

Будьте обережні, вносячи зміни у файл таким чином. Деякі контейнери файлів містять більше інформації, як-от розташування, теги або примітки, які не відображатимуться в новому типі файлу.

Розуміння програми Файли

iPhone і iPad принципово відрізняються від обчислювальних платформ Mac, тому користувачі не повинні очікувати, що Files матиме однакові функції з Finder. Однією з найбільш істотних відмінностей між ними є доступ до системних файлів.

На Mac користувачі можуть занурюватися в приховані файлові структури та змінювати програмні файли для встановлених програм або навіть системні файли, невід'ємні для нормальної роботи операційної системи. На iPad і iPhone єдине, до чого користувачі мають доступ, – це файли, створені користувачами.

Apple взагалі цього не відкривала, і користувачі не повинні очікувати, що ця філософія зміниться найближчим часом. Натомість оновлення Files зосереджені на покращенні якості життя, що полегшує керування файлами.

Знайдіть файли на своєму iPhone або iPad у програмі Файли

Дізнайтеся, як знаходити та редагувати файли за допомогою програми Файли.

Як знайти свої файли в програмі Файли

1. На своєму iPhone або iPad відкрийте програму Файли. Не бачите синього значка програми Файли на головному екрані? Просто проведіть пальцем униз і знайдіть програму Файли.

2. У програмі Файли знайдіть потрібний файл або знайдіть його.

Використовуйте програму Файли для керування файлами

На вашому iPhone або iPad ви можете зберігати та редагувати свої файли в програмі Файли.

Які типи файлів зберігаються в програмі Файли

– У папці «На моєму [пристрої]» можна знайти файли, що зберігаються локально на пристрої, який ви використовуєте.

– У папці iCloud Drive можна знайти файли та папки, які синхронізуються з iCloud Drive, зокрема сторінки, номери, документи Keynote тощо.

– Ви також можете знайти файли в інших хмарних службах і програмах, які ви підключаєте, наприклад Box, Dropbox, OneDrive, Adobe Creative Cloud, Google Drive тощо.

Програма Файли не містить вашої бібліотеки фотографій. Але ви можете зберігати фотографії в програмі Файли.

Папки та каталоги zip, захищені паролем, не підтримуються в програмі Файли.

Як редагувати файли в програмі Файли

Щоб відредагувати файл, просто натисніть, щоб відкрити його. Залежно від типу файлу ви можете:

– Використовуйте Markup для малювання, додавання тексту, додавання підпису тощо.

– Відредагуйте файл у Pages, Keynote, Numbers та інших програмах.

Ви також можете видалити файли з програми Файли. Зміни файлів, що зберігаються в iCloud Drive або інших хмарних службах, автоматично оновлюються на всіх ваших пристроях.

3.2 Розробка структурної схеми

Розроблювальна система диспетчеризації файлів у мобільній операційній системі iOS 16 – це те, як ви отримуєте доступ до файлів, переглядаєте та керуєте ними в iOS і iPadOS. Це також чудовий спосіб підключення до онлайн-служб, мережевих серверів і зовнішніх дисків.

Програма Apple Files дозволяє переглядати та керувати файлами, що зберігаються в онлайн-сервісах, таких як iCloud Drive, Box, Dropbox, Google Drive і Microsoft OneDrive, усе в одному місці. Ви також можете переглядати свої файли безпосередньо на своєму iPhone або iPad і виконувати з ними різні команди.

З оновленням до iOS 16 і iPadOS 16 Apple доповнила програму Files новими цікавими функціями. Ви можете сортувати файли за групами, легше виконувати багатозадачність із кількома вікнами та вибирати кілька файлів, перетягуючи їх за допомогою зовнішньої миші.

Щоб переконатися, що ви можете використовувати найновіші функції, перейдіть до Налаштування > Загальні > Оновлення ПЗ. Ваш пристрій повідомить вам, що програмне забезпечення оновлено, або запропонує завантажити останнє оновлення.

Давайте перевіримо програму Файли, щоб дізнатися, як ви можете працювати зі своїми документами, фотографіями та файлами.

Підключіть служби до програми Files

Коли ви вперше запускаєте Files, вам потрібно буде ввімкнути служби, які ви використовуєте, і підключитися до них. Торкніться піктограми з крапкою вгорі та виберіть у меню «Редагувати», а потім увімкніть будь-які онлайн-локації, які ви хочете додати. Ви також можете натиснути на піктограму гамбургера поруч із улюбленими послугами та перетягнути кожну з них угору чи вниз, щоб відсортувати їх так, як вам потрібно.

Автентифікація послуги

Торкніться назви служби, яку потрібно завантажити, і ввійдіть. Коли ви це робите вперше, для перегляду елементів у цій папці потрібна автентифікація. У деяких випадках ви можете отримати повідомлення автентифікації, що означає, що вам потрібно спочатку відкрити мобільну версію програми, щоб використовувати її у Файлах. Робіть це для кожного сайту, який ви додаєте.

Підключіться до зовнішнього накопичувача

Ви можете підключити свій пристрій до USB-накопичувача, SD-карти або зовнішнього жорсткого диска. Для цього вам потрібно фізично підключити накопичувач до вашого iPhone або iPad за допомогою відповідного адаптера та кабелю. Потім ви зможете отримати доступ до цього диска та його вмісту через програму Файли.

Якщо у вас є мережевий сервер або NAS, ви можете підключитися до нього за допомогою свого пристрою та отримати до нього доступ через програму Файли. Просто переконайтеся, що він підтримує та ввімкнув SMB (Server Message Block), універсальний мережевий протокол, який дозволяє різним системам отримувати доступ до тих самих файлів і спільно використовувати їх.

Підключення до зовнішнього накопичувача

Щоб налаштувати це в програмі «Файли», торкніться піктограми з крапкою у верхньому правому куті, потім торкніть команду «Підключитися до сервера» та введіть назву або IP-адресу сервера. (Наприклад, щоб отримати доступ до мого Synology NAS, мені потрібно було ввести smb:// , потім назву NAS, а потім .local, як у smb://SynologyNAS.local.) Потім введіть ім'я користувача та пароль для цього пристрою.

Спочатку я не міг підключитися до NAS. На корисному онлайн-форумі я виявив, що мені потрібно встановити максимальний протокол SMB на SMB3. Якщо у вас виникли проблеми з підключенням до сервера або NAS через програму «Файли», вам може знадобитися зв'язатися з постачальником по допомогу.

Підключіться до зашифрованого диска

Програма Files також сумісна із зашифрованим диском, хоча наразі підтримуються лише диски, відформатовані за допомогою APFS. Щоб спробувати це, підключіть зашифрований пристрій зберігання даних або підключіться до зашифрованої спільної мережі. Виберіть диск і торкніться посилання «Заблокувати». Введіть пароль для диска, а потім виберіть Розблокувати, щоб отримати доступ до диска.

Пошук і перегляд файлів

Пошук файлів

Ви можете знайти файли в різних місцях. У полі пошуку вгорі введіть слово або фразу для потрібного файлу. Програма сканує ваші різні місця та відображає список результатів пошуку у відповідь.

Якщо ви часто користуєтеся одними і тими самими папками, позначте їх як уподобання, щоб швидко отримати до них доступ у майбутньому. Натисніть папку та виберіть у меню «Вибране». Після цього на лівій бічній панелі з'явиться розділ «Вибране». Щоб позначити інші папки як вибрані, перетягніть їх у розділ «Вибране».

Позначте послугу як улюблену

Нещодавно доступні файли можна швидко знайти всередині програми. Торкніться запису «Останні» на бічній панелі, щоб переглянути нещодавно додані, переміщені або переглянуті файли.

Ви також можете отримати доступ до файлів, завантажених у Safari, через програму Файли. Завантажте PDF-файл, аудіофайл або документ із Safari, і папка «Завантаження» з'явиться в програмі «Файли». Торкніться «На моєму iPhone» або «На моєму iPad», і ви побачите папки для інших програм і служб на вашому пристрої, а також папку «Завантаження». Відкрийте цю папку, щоб отримати доступ до файлів, завантажених у Safari.

Доступ до завантаженого файлу

Торкніться певної файлової служби, щоб переглянути папки та файли, які в ній зберігаються. Торкніться файлу, щоб відкрити його, і програма завантажить і відобразить файл. Таким чином можна переглядати документи, файли PDF, зображення, аудіофайли та відео. Фотографії та інші статичні зображення можна навіть редагувати за допомогою вбудованих інструментів малювання та розфарбовування.

Керуйте своїми файлами

Ви можете керувати основними файлами, копіюючи, переміщуючи або видаляючи файли. Відкрийте одну зі своїх онлайнових служб зберігання, щоб переглянути збережені папки та файли. Щоб виконати команду для одного файлу, натисніть його ескіз. З меню ви можете копіювати, дублювати, переміщувати, перейменовувати, переглядати, позначати тегами, ділитися або видаляти файл.

Щоб виконати команду для кількох файлів, торкніться посилання

«Вибрати» у верхньому правому куті на iPad (кнопка прихована за піктограмою з трьома крапками на iPhone), а потім торкніться кожного файлу, який потрібно включити. У нижній частині екрана відображаються посилання для спільного використання, дублювання, переміщення та видалення вибраного файлу. Торкніться «Більше», щоб побачити додаткові команди, наприклад «Завантажити» та «Стиснути».

Переглянути файл

Є також команди, коли ви переглядаєте файл. У вікні перегляду натисніть стрілку вниз біля назви файлу. Залежно від типу файлу ви можете потенційно зберегти, скопіювати, перемістити, перейменувати, експортувати або надрукувати його.

Виберіть кілька файлів

Оскільки ваш iPad працює із зовнішньою мишею, ви можете вибрати кілька файлів одночасно, не використовуючи опцію «Вибрати». Переконайтеся, що вашу мишу підключено до вашого iPad, а потім клацніть і перетягніть рядок файлів, щоб вибрати їх усі.

Програма Файли дозволяє стискати та розтискати файли та папки. Натисніть на файл і виберіть «Стиснути» у спливаючому меню, щоб створити його ZIP-архів. Ви також можете стискати кілька файлів разом. Якщо у вас є заархівований файл, який потрібно відкрити, просто торкніться його, щоб розпакувати, або натисніть на файл і виберіть у меню «Розпакувати».

Видалити файли

Якщо ви хочете видалити файл, натисніть на нього, щоб відкрити спливаюче меню, а потім виберіть команду «Видалити». Будь-який видалений файл можна повернути, торкнувшись запису «Нещодавно видалені». Це покаже вам усі нещодавно видалені файли в різних службах зберігання файлів.

Торкніться посилання «Вибрати», а потім виберіть файл, який потрібно відновити. Торкніться посилання «Відновити», щоб повернути видалений файл до служби зберігання.

Позначення та сортування файлів

Перегляд інформації про файли

Програма Files дозволяє переглядати ключові деталі та навіть позначати кожен файл. Натисніть на файл і виберіть у меню « Інформація», щоб переглянути тип файлу, розмір, дату й час створення, автора та інші відомості. За допомогою фотографії ви можете побачити розмір, роздільну здатність, час витримки, фокусну відстань, швидкість ISO тощо.

Позначити файл тегом

Ви можете додати тег, щоб класифікувати файл і допомогти вам його знайти. Натисніть файл і виберіть «Тег» або проведіть пальцем униз донизу екрана «Інформація» та торкніться « Додати теги». Потім ви можете додати тег із наявного кольорового списку або натиснути «Додати новий тег», щоб створити новий на основі певних критеріїв. Коли закінчите, торкніться Готово.

Ви можете сортувати файли в папці за різними критеріями. Щоб спробувати це, відкрийте папку, торкніться значка кола з трьома крапками на iPhone та значка чотирьох квадратів на iPad. У меню виберіть критерії сортування, наприклад ім’я, тип, дата, розмір або теги.

Сортування та групування файлів

Щоб згрупувати файли за певними критеріями, натисніть команду «Параметри перегляду». Потім ви зможете впорядкувати файли в групи за типом, датою, розміром або статусом спільного доступу. Ви також можете вибрати відображення розширень для файлів.

Багатозадачність із кількома вікнами

Запустіть режим розділеного перегляду

Ваш iPad має низку можливостей багатозадачності. Ви можете легше переглядати та працювати з файлами, коли у вас одночасно відкрито кілька вікон. Відкрийте папку в програмі «Файли» та торкніться піктограми з крапкою у верхній частині екрана, щоб відкрити панель інструментів багатозадачності. Натисніть команду Split View, щоб запустити режим Split View.

Перегляньте файл на окремій панелі

Потім ви можете відкрити другий екземпляр програми «Файли» та переглянути іншу папку на іншій панелі. Якщо ви хочете відкрити певний файл, торкніться піктограми з крапкою та виберіть « Розділений перегляд» або «Пересунути». Потім ви можете переглянути файл на окремій панелі.

Структурна схема розробленої системи зображена на рисунку 3.1. На ній показано структуру розробленого програмного забезпечення диспетчеризації файлів у мобільній операційній системі iOS 16. Вона складається з наступних основних структурних блоків:

– Система диспетчеризації файлів у мобільній операційній системі iOS 16.

– Комунікації.

– Операційна система Apple iOS.

– Вбудовані додатки.

– Пам’ять.

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.

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).

Шар мови С 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.

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**.**

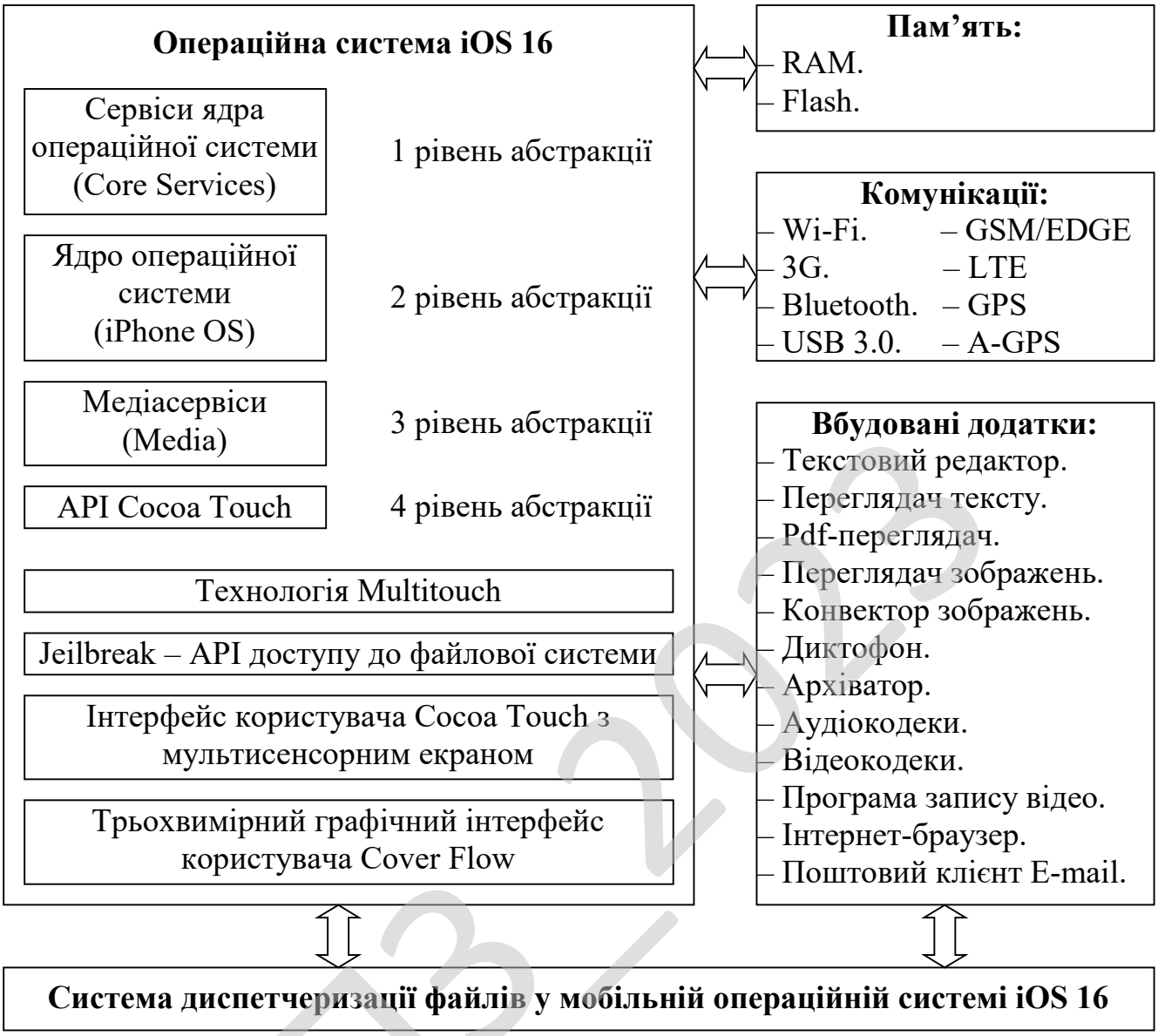

Рисунок 3.1 – Структурна схема системи

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.

Jaibreak

Джейлбрейк – це процес, що дозволяє розкрити справжній потенціал iPhone та прибрати обмеження, встановлені в операційній системі iOS. Це робиться за допомогою установки "Менеджера пакетів", такого як Cydia або Sileo. Також джейлбрейк дає можливість встановлювати сторонні програми та твікі для налаштування та покращення вашого пристрою.

Джейлбрейк легальний. У 2012 році Бібліотека Конгресу визнала його юридичним винятком із DMCA (Закон про захист авторських прав у цифрову епоху), що робить його законним.

Менеджер пакетів такий як Sileo та Cydia встановлюється при джейлбрейку на iPhone. Він дозволяє встановлювати сторонні програми та налаштування на iOS. Він нагадує неофіційний магазин додатків.

Cydia – це менеджер пакетів (неофіційний магазин додатків) для пристроїв iOS, який дозволяє завантажувати налаштування та програми на iPhone та iPad. Щоб завантажити Cydia на телефон, необхідно виконати джейлбрейк пристрою за допомогою перерахованих вище інструментів.

Навіщо потрібний джейлбрейк для встановлення Cydia?

Apple сильно обмежує той контент, який вони дозволяють вам завантажувати. Якщо програма не з офіційного магазину програм, то Apple не хоче, щоб вона у вас з'явилася. Cydia – один із способів завантажувати неофіційні програми та інший контент на ваш пристрій. Цей магазин програм включений у кожен джейлбрейк ; і ви не зможете встановити його окремо.

Cydia пропонує кілька функцій, які без джейлбрейку недоступні.

– Завантаження тем і налаштувань, які змінюють зовнішній вигляд пристрою.

– Завантаження програм та налаштувань, які привносять додаткові функції та покращення роботи iOS.

– Завантаження програм та налаштувань, які забезпечують додаткові функції для стандартних програм iOS.

– Завантаження безлічі рінгтонів, шпалер та інших корисних покращень, недоступних у стандартній iOS.

Все це робить завантаження Cydia справою, що стоїть.

Cydia не може бути встановлена якщо заздалегідь не зробити джейлбрейк.

Так, Cydia – це повністю безкоштовний магазин додатків для iPhone, який доступний для завантаження за допомогою будь-якого з наведених вище інструментів для джейлбрейка.

Так, Cydia є абсолютно безпечною. Тим не менш, ви повинні використовувати лише авторитетні джерела репозитаріїв для Cydia і завантажувати вміст звідти. Сторонні чи не перевірені джерела можуть містити шкідливе програмне забезпечення.

Джейлбрейк дає вам доступ до безлічі додаткових функцій та покращень, таких як:

– Отримання більш глибокого доступу до iOS.

– Доступ до менеджерів пакетів, таких як Cydia або Sileo.

– Встановлення тем, які змінять вигляд вашого пристрою.

– Встановлення установок, які змінять поведінку Вашого пристрою.

– Встановлення стороннього контенту, недоступного в інших місцях.

Налаштування та програми Cydia безкоштовні, однак деякі з них вимагають невеликої оплати.

Альтернативи Cydia є і можуть навіть не вимагати робити джейлбрейк для вашого пристрою. Вони прості у завантаженні та зручні у використанні. Ми перерахували деякі з них на цій сторінці.

Щоб видалити Cydia, ви можете або відновити ваш пристрій за допомогою iTunes або скористайтеся програмою Cydia Eraser. Cydia Eraser – це корисний інструмент, який видаляє Cydia з iPhone без оновлення поточної прошивки iOS. Це означає, що якщо ви хочете, то можете повторно зламати свій пристрій і встановити Cydia пізніше.

Репозиторій або сховище – це як база даних, повна налаштувань та програм, які ви можете додати до менеджера пакетів (наприклад, у Cydia або Sileo) на своєму зламаному пристрої. Менеджери пакетів зазвичай постачаються з кількома попередньо встановленими джерелами репозиторіїв, але ви можете додати додаткові джерела, які дадуть вам доступ до більшої кількості налаштувань та програм.

Твік (або налаштування) – це програма, яка змінює зовнішній вигляд пристрою або певних його частин. Установки зазвичай не мають піктограми програми, на яку ви можете натиснути. Натомість вони зібрані у програмі «Параметри», де їх можна налаштувати.

Коли ваш пристрій відновлює роботу, домашня сторінка та інтерфейс iOS перезапускаються. Більшість налаштувань вимагають, щоб ви “переналаштували” свій пристрій, щоб зміни набули чинності без перезавантаження телефону. Вам також не потрібно встановлювати заново джейлбрейк, якщо ваш пристрій відновиться, і менеджери пакетів (тобто Cydia і Sileo) залишаться активними.

Режим відновлення – це свого роду захист вашого iPhone або iPad, який дозволяє виконати нове встановлення операційної системи iOS. Режим

відновлення – це корисна функція, яка використовується при відновленні операційної системи iPhone, якщо якась її частина пошкоджена або запускається неправильно. Це може статися під час невдалої спроби джейлбрейка.

Режим DFU дозволяє взаємодіяти з операційною системою без завантаження iOS чи самого завантажувача. DFU означає оновлення прошивки пристрою, але його не слід плутати з режимом відновлення, оскільки у функціональному плані він відрізняється. DFU зазвичай використовується як останній засіб, якщо щось піде не так на вашому iPhone або iPad. Деякі інструменти також використовують його як частину процесу джейлбрейку вашого пристрою.

3.3 Розробка функціональної схеми

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

– Блок завантаження/вивантаження файлів.

– Відкриття файлів.

– Система диспетчеризації файлів у мобільній операційній системі iOS 16.

– Параметри системи диспетчеризації файлів у мобільній операційній системі iOS 16.

Розглянемо більш детально кожний з блоків розробленої програми.

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 диспетчеризації файлів у мобільній операційній системі iOS 16.

– Відкриття файлів сторонніми додатками.

Система диспетчеризації файлів у мобільній операційній системі iOS 16 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.

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

Рисунок 3.2 – Функціональна схема системи

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. Після початку роботи розробленого ПЗ ми потрапляємо до головного блоку системи звідки через ланку дій відбувається наступне:

– Інтерфейс ПЗ.

– Налаштування ПЗ.

– Обробник помилок.

– Моніторинг накопичувача.

– Форматування накопичувача.

– Виведення властивостей накопичувача.

– Виведення розміру зайнятої та вільної пам’яті.

– Виведення розміру накопичувача.

– Виведення списку файлів та каталогів.

– Вибір файлів/каталогів.

– Передача файлу на зовнішній пристрій.

– Вибір зовнішнього пристрою та підключення до нього .

– Перейменування файлу.

– Створення файлу.

– Архівування файлу.

– Переміщення/копіювання/видалення файлу.

– Виведення інформації про файл/каталог.

Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі. Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

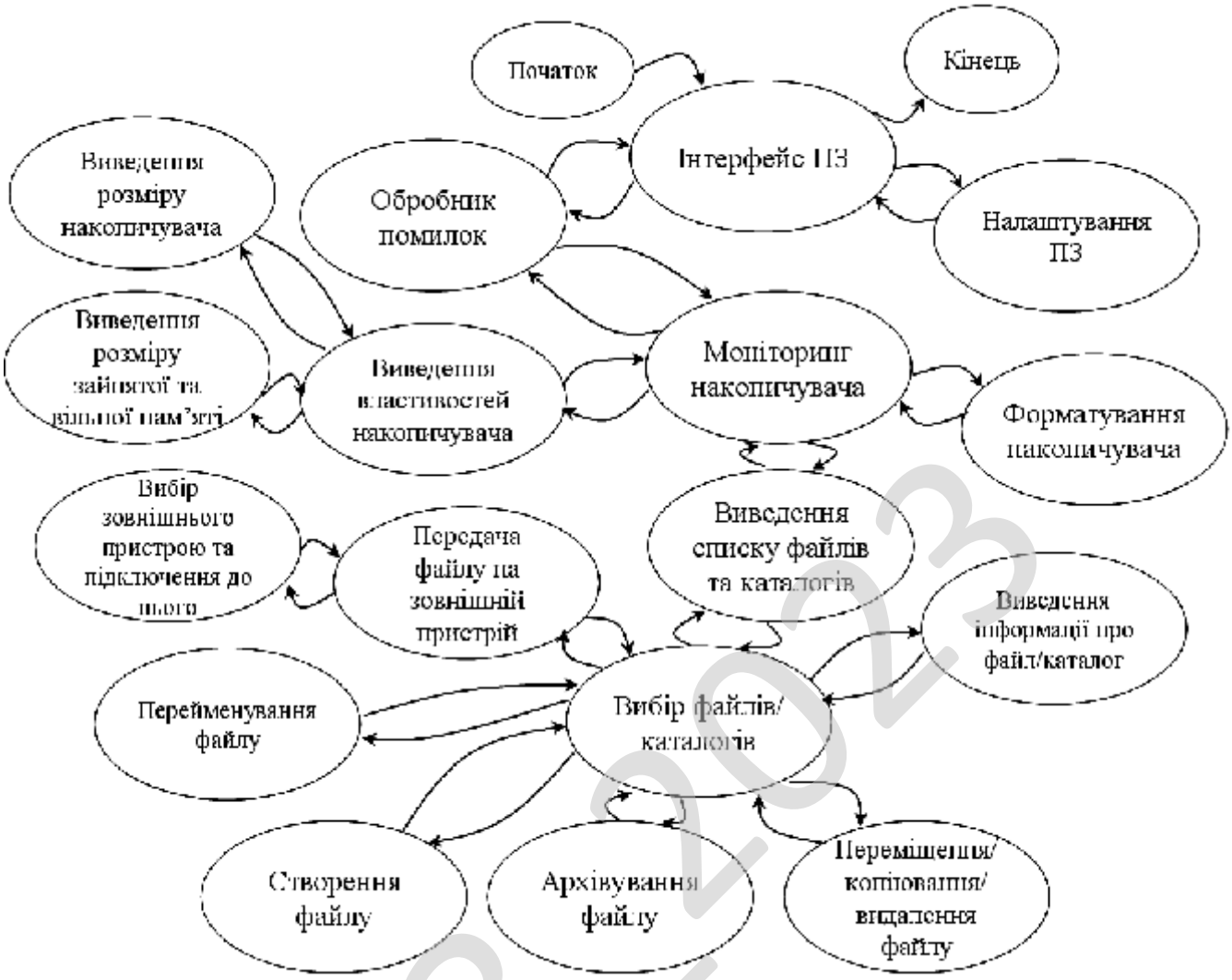

Рисунок 3.3 – Діаграма взаємодії процесів

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

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем. На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми. З якої видно що робота основної програми складається з декількох етапів.

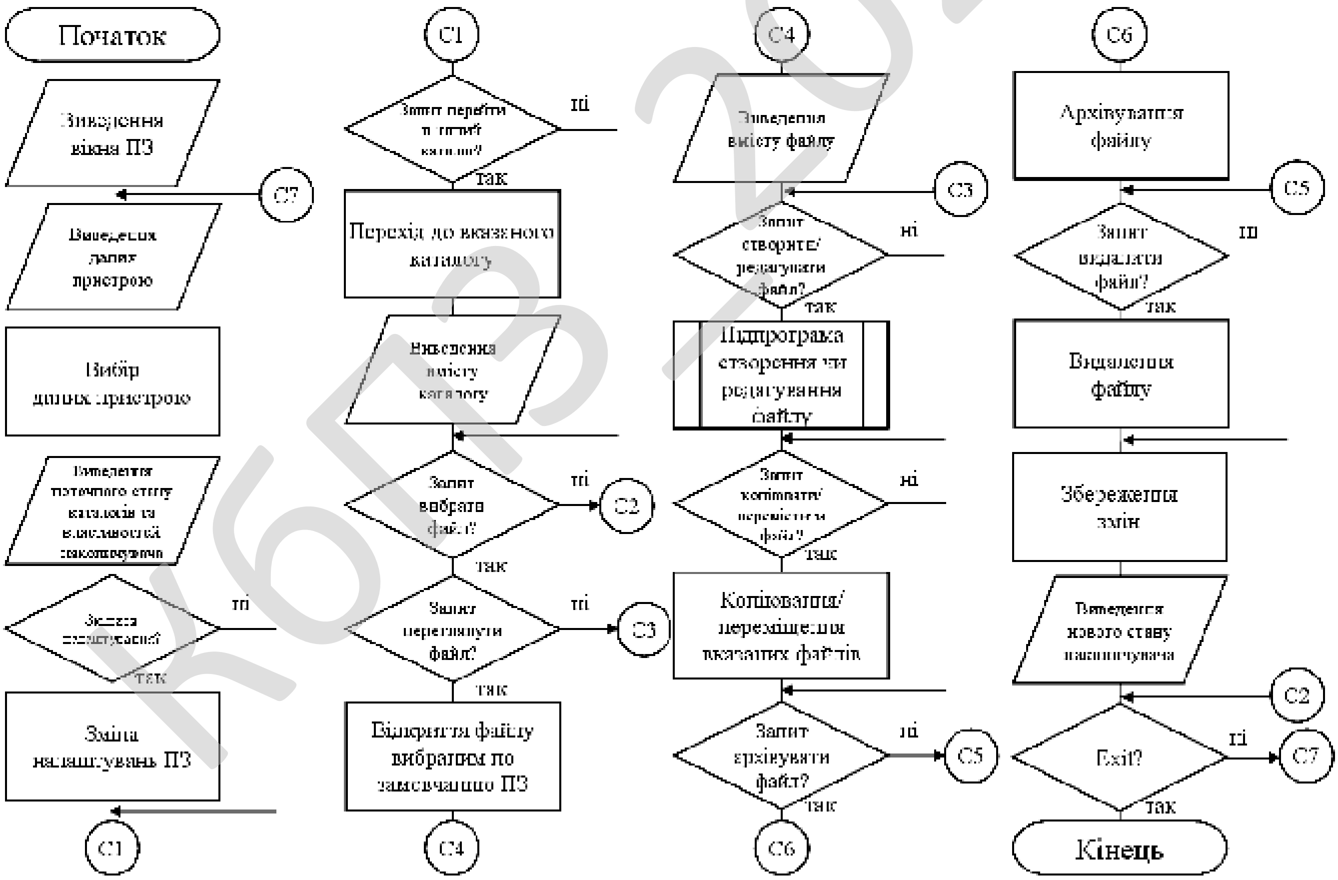

Рисунок 4.1 – Блок схема основної програми

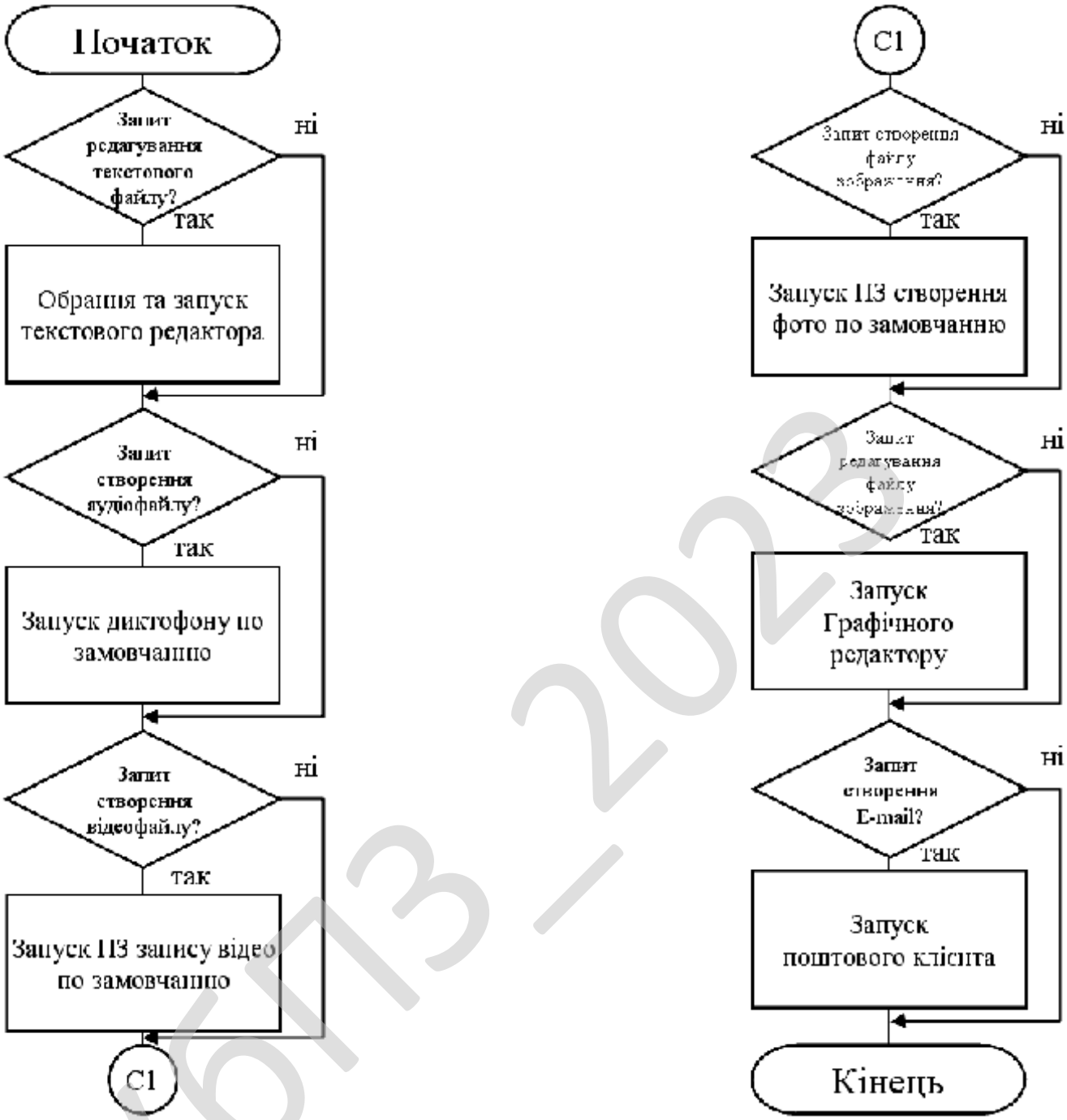

Рисунок 4.2 – Блок схема підпрограми

Початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ.

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

Опис алгоритмів функціонування системи

Опишемо та розглянемо, як реалізуються елементи програмного забезпечення системи диспетчеризації файлів у мобільній операційній системі iOS 16.

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.

Опис покажчиків та типів які використовуються

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).

Опис визначених 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 С. 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 С, 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).

Опис розширених класів

Однієї з речей, прямих аналогів якої немає в С++, 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 С++. 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.

Опис об'єктної моделі

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.

Опис протоколів

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.

Опис неформальних протоколів

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.

Опис 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.

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.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм ДСТУ 28147:2009, що є класичним алгоритмом симетричного шифрування на основі мережі Фейстеля (рисунок 4.3). Даний алгоритм шифрує інформацію блоками по 64 біта (такі алгоритми називаються "блоковими"). Зміст мережі Фейстеля полягає в тому, що блок шифруемої інформації розбивається на два або більше субблоків, частина яких обробляється за певним законом, після чого результат цієї обробки накладається (операцією побітового додавання за модулем 2) на необроблювані субблоки. Потім субблоки міняються місцями, після чого обробляються знову й т.д. певне для кожного алгоритму число раз – раундів.

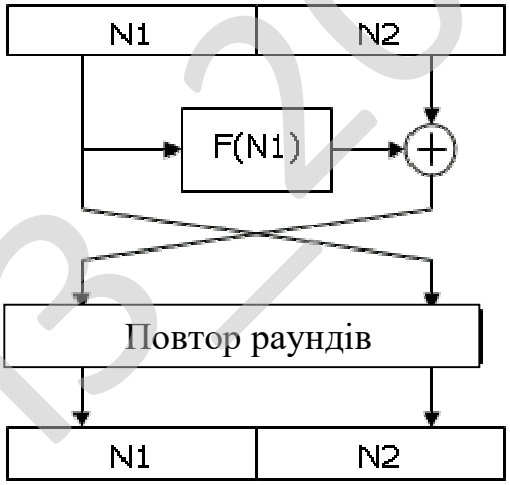

Рисунок 4.3 – Мережа Фейстеля

Основна відмінність алгоритмів симетричного шифрування друг від друга складається саме в різних функціях обробки субблоків. Дана функція часто називається "основним криптографічним перетворенням", оскільки саме вона несе основне навантаження при шифруванні інформації. Основне перетворення алгоритму ДСТУ 28147:2009 є досить простим, що забезпечує високу швидкодію алгоритму; у ньому виконуються наступні операції (рисунок 4.4).

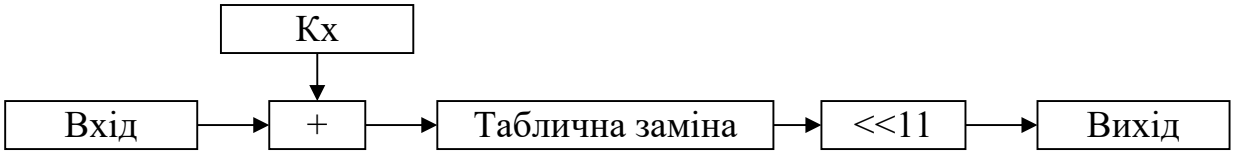

Рисунок 4.4 – Основне перетворення алгоритму ДСТ 28147:2009

1. Додавання субблоку з певним фрагменом ключа шифрування за модулем 2^{32}. Кх – це 32-бітна частина ("підключ") 256-бітного ключа шифрування, якому можна представити як конкатенацію 8 підключів: К = $К_0К_1К_2К_3К_4К_5К_6$ $К_7$. Залежно від номера раунду й режиму роботи алгоритму (про їх – нижче), для даної операції вибирається один з підключів.

2. Таблична заміна. Для її виконання субблок розбивається на 8 4-бітних фрагментів, кожний з яких прогоняється через свою таблицю заміни. Таблиця заміни містить у певній послідовності значення від 0 до 15 (тобто всі варіанти значень 4-бітні фрагменти даних); на вхід таблиці подається блок даних, числове подання якого визначає номер вихідного значення. Наприклад, подається значення 5 на вхід наступної таблиці: "13 0 11 74 91 10 143 5 122 15 8 6". У результаті на виході виходить значення 9 (оскільки 0 заміняється на 13, 1 – на 0, 2 – на 11 і т.д.).

3. Побітове циклічне зрушення даних усередині субблока на 11 біт уліво.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на розділи та дозволяє визначити елемент файлової системи та проводити роботу з обраним елементом файлової системи.

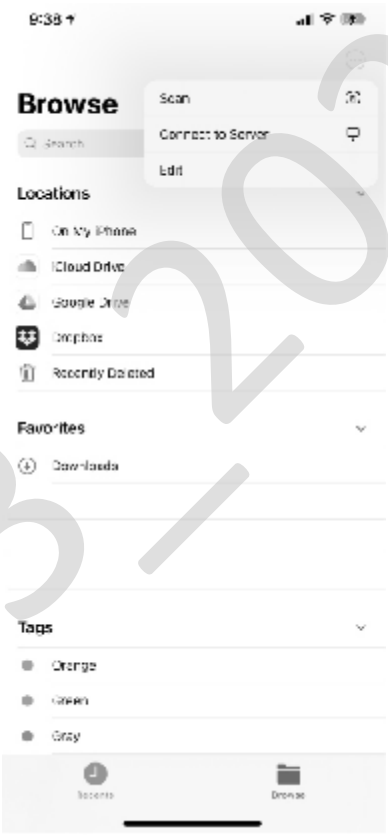

Рисунок 5.1 – Головне вікно програми

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

Обрано умови розповсюдження – Freeware.

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

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

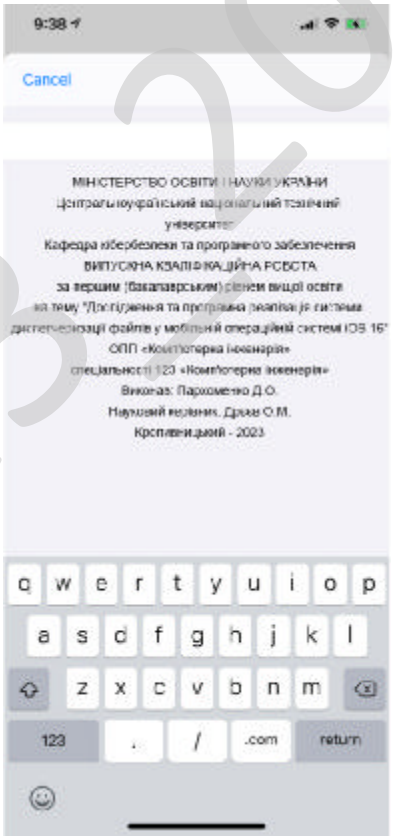

Рисунок 5.2 – Авторське право

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи диспетчеризації файлів у мобільній операційній системі iOS 16.

Метою розробки є дослідження та програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16.

Об'єктом дослідження є процес диспетчеризації файлів у мобільній операційній системі iOS 16.

Предметом дослідження є методи диспетчеризації файлів у мобільній операційній системі iOS 16.

Методи дослідження базуються на методах комп'ютерної інженерії, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод диспетчеризації файлів у мобільній операційній системі iOS 16.

– Розроблено вітчизняний продукт диспетчеризації файлів у мобільній операційній системі iOS 16, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

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

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

В магістерській роботі було проведене дослідження та виконана програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16.

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	120
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

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

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	120000
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: *Wi* – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

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

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

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

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

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

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

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

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

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

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

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

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

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 50 = 84 \text{ люд/день.}$$

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	84	Ф 7.1-7.4
Впровадження	13	Д13
Всього	125	–

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.

$$Ч = \frac{125 \cdot 1}{60 - 5} = 2,3 \text{ ставки.}$$

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	6	540	9
Монітор	60	6	360	6
Клавіатура	30	6	180	3
Маніпулятор «мишка»	30	6	180	3
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор-маршрутизатор	30	1	30	0,5
Кабельні господарства ЛОМ на 1 м.п.	2,5	260	650	10,83
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	37,99

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

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

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

$$Ф_{др}^{с} = \frac{38 \cdot 3}{1{,}2} = 95 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

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

$$Ч_{ел} = 95/(60 \cdot 8) = 1{,}2 \text{ ставки.}$$

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

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

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

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

Продовження таблиці 7.4

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

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (IT-менеджер)	0,5	18100	27150
Продакт-менеджер	0,25	10000	7500
Інженер-програміст	2,3	16000	110400
Інженер-електронщик	0,2	10000	6000
Інженер-системотехнік	0,25	10000	7500
Адміністратор мережі	0,5	10000	15000
Системний програміст	0,25	10000	7500
Дизайнер WEB	0,25	10000	7500
Інженер-верстальник	0,25	10000	7500
Бухгалтер-економіст	0,5	10000	15000
Всього за період розробки	$R_{сп}$ = 5,25	-	$Ф_{роб}$ = 211050

Розрахуємо середньоденну зарплату одного виконавця:

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

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

$$З_{сд} = \frac{211050}{5,25 \cdot 60} = 670 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

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

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

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

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

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

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

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

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

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

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

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

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

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

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу фірми Brain за 26.10.23 – джерело http: //brain.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		10947
Системний блок		7347
Процесор	INTEL Core™ i3 10105 (BX8070110105) 1200, 4 ядра, 8 потоків, 3.7 GHz, 4.4 GHz, TDP - 65 Вт, 14nm, - 6 MB Intel Smart Cache, 8 GT/s, Comet Lake, BOX	-
Системна плата	GIGABYTE H470M H сокет - 1200, DDR4, - 64 ГБ, - 3200 MHz, LAN - 1 Гбит/с, D-Sub (VGA), HDMI, 1 x M.2 2280, 4 x Sata 6.0 Gb/s, Micro-ATX	-
Відеокарта	Intel UHD Graphics 630	-
Жорсткий диск	SSD 2.5" 256GB Mibrand (MI2.5SSD/CA256GBST) 256 GB, 3D TLC NAND, 2.5", SATA III (6Gb/s)	-
Оперативна пам’ять	DDR4 8GB 2666 MHz eXceleram (E408266A) DDR4, 8 ГБ, В наборі - 1	-
DVD-привод	-	-
Корпус	ATX Middle Tower FOXCONN Pro, 3GTLA-489, PSU 350W(FSP Brand: ATX-350PNR, 12cm), black, (front bezel – black+light silver; body material – 0.6mm), 80mm fan (rear), 2xUSB2.0/AUDIO/MIC, Air Duct, Tool-less chassis design,Thermally Advantaged Chassis	-

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Кулер	–	–
Кардрідер внутрішній	USB 3.0 Card reader. 3.5", 2*USB3.0 +AUDIO+1394, multi: All Type Cards, black	-
інше	Клавіатура, мишка	Подару-нок
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1, 170/160, D-SUB, Wide)	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

Таблиця 7.7 – Балансова вартість обчислювальної техніки

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

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

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

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

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

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

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

$$З_о = 670 \cdot 125/120 = 698 \text{ грн.}$$

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, %.

$$З_д = 698 \cdot 10 \cdot 0{,}01 = 70 \text{ грн.}$$

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

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

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

$$С_{оц} = 0{,}01 \cdot 22(698+70) = 288 \text{ грн.}$$

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

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

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 698 \cdot 15 \cdot 0{,}01 = 105 \text{ грн.}$$

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

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

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

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 0,5 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=210 грн., визначаємо вартість паперу за період розробки:

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

$$З_{М1} = 210 \cdot 0{,}5 = 105 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 60):

$$З_{М2} = \sum Ц_д\ , \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 24 грн./шт., DVD-R box – 35 грн./шт.

$$З_{М2} = 60 \cdot 24 = 1440 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з.., \quad (7.18)$$

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

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

$$З_М = (105 + 1440 + 1702)/120 = 27 \text{ грн.}$$

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, %.

$$О_п = 698 \cdot 15 \cdot 0{,}01 = 105 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 120 прим.):

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

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

$$A_м = 190286 \cdot 3/(120 \cdot 12) = 396 \text{ грн.}$$

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

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

$$C_п = 698+70+288+105+27+105+396 = 1689 \text{ грн.}$$

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

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн
1. Основна зарплата виконавців	$З_о$	698
2. Додаткова зарплата виконавців	$З_д$	70
3. Відрахування на соціальні потреби	$C_{оц}$	288
4. Загальногосподарські витрати	$Г_{осп}$	105
5. Витрати на матеріали	$З_М$	27
6. Освоєння нових операційних систем, мов програмування	$О_п$	105
7. Амортизація основних фондів	$A_м$	396
8. Повна собівартість програмного забезпечення	$C_п$	1689
9. Плановий прибуток	$П_р$	845
10.Ціна підприємства $Ц_п = C_п + П_р$	$Ц_п$	2534
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	506,8
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	3040,8

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

Для даного програмного забезпечення рівень рентабельності складає 50%.

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

де: P_n – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 50 \cdot 1689 = 845 \text{ грн.}$$

7.6 Визначення об'єму капітальних вкладень у споживача програмної продукції

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

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

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	3041
Всього капітальних витрат	–	3041

7.7 Визначення експлуатаційних витрат

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

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на обслуговування системи	$З_р$	36905	29524
2. Витрати на електроенергію	$З_{ел}$	439	251
3. Витрати на амортизацію	$З_{ам}$	0	760
Всього витрат за рік	$І$	37344	30535

Витрати на обслуговування роботи системи:

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

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

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

Після купівлі нового програмного забезпечення кількість годин на обслуговування системи зменшилось з 250 год до 200 год на рік.

$$З_{р\,баз} = 250 \cdot 110 \cdot 1{,}1 \cdot 1{,}22 = 36905 \text{ грн},$$

$$З_{р\,нов} = 200 \cdot 110 \cdot 1{,}1 \cdot 1{,}22 = 29524 \text{ грн}.$$

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	–	3041	–	760,25
Всього відрахувань	-	–	3041	–	760,25

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

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

$$З_{ел\ баз} = 0{,}545 \cdot 350 \cdot 2{,}3 = 439 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}545 \cdot 200 \cdot 2{,}3 = 251 \text{ грн.}$$

7.8 Визначення економічної ефективності програмної продукції

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

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

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

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

$$Ев = (2534 - 1689) \cdot 120 - (0{,}05 \cdot 1408000 + 0{,}4 \cdot 199177 +$$
$$+0{,}25 \cdot 33190 + 0{,}1 \cdot 120000) \cdot 3/12 = 53828{,}5 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

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

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

$$T_в = \frac{1760367}{(2534 - 1689) \cdot 120 \cdot 12 / 3} = 4 \text{ роки}$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

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

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

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

$$E_{сп} = (37344\text{-}30535)\text{-}0{,}25 \cdot 3041 = 6049 \text{ грн.}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	120
2. Повна собівартість розробленої програми	Грн.	1689
3. Ціна розробленої програми	Грн.	2534
4. Плановий прибуток від реалізації розробленої програми	Грн.	845
5. Рентабельність програмної продукції	%	50
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1760367
7. Загальний прибуток від реалізації програмної продукції	Грн.	101400
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	53828,5
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Рік	4
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	3041
11.Величина економічного ефекту у користувача програмної продукції	Грн.	6049
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Року	0,45

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

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

$$T_{сп} = \frac{3041}{37344 - 30535} = 0,45 \ року .$$

7.9 Висновки

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

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Техніка безпеки – це система правил і заходів, які допомагають запобігти травмам, хворобам і аваріям на робочому місці або в повсякденному житті. Знання техніки безпеки дуже важливе, бо воно рятує життя і здоров'я людей. Наприклад, якщо людина працює на шахті, то вона повинена знати правила безпеки у вугільних шахтах, щоб не потрапити під обвал або не підірватися на міні. Або якщо людина хоче навчитися програмувати, то вона повинена дотримуватися гігієнічних вимог і не сидіти за комп'ютером занадто довго, щоб не зашкодити своїм очам і спині.

Охорона праці та здоров'я у сфері ІТ – це комплекс заходів, які спрямовані на забезпечення безпечних і здорових умов праці для працівників, які використовують інформаційні технології, а також на запобігання травматизму, професійним захворюванням і стресу.

Охорона праці та здоров'я у сфері ІТ включає такі напрями, як:

– Ергономіка – це наука про адаптацію робочого середовища до фізичних і психологічних особливостей людини[2]. Ергономіка вимагає врахування таких факторів, як розміри, форми, кольори, освітлення, шум, температура, вологість, вентиляція, постава, рухи, пози, втома, навантаження на очі та ін. Егономіка допомагає покращити комфорт, продуктивність і задоволення працівників.

– Комп'ютерна безпека – це захист комп'ютерних систем і даних від несанкціонованого доступу, зміни, знищення або блокування. Комп'ютерна безпека вимагає використання антивірусних програм, фаєрволів, паролей, шифрування, резервного копіювання та інших технологій. Комп'ютерна безпека допомагає запобігти крадіжці, шпигунству, шантажу, саботажу та іншим загрозам.

– Соціальна взаємодія – це процес спілкування між людьми у робочому колективі або через мережеві сервіси. Соціальна взаємодія вимагає дотримання правил етикету, поваги, толерантності, співробітництва та конструктивного діалогу. Соціальна взаємодія допомагає покращити настрій, мотивацію, комунікацію та творчий потенціал працівників.

Правила охорони праці і здоров'я для програмістів:

– Регулярно роби перерви в роботі. Вставай із-за столу і розминай м'язи.

– Налаштуй яскравість і контрастність монітору так, щоб не напружувати очі.

– Використовуй ергономічну мишку і клавіатуру, які зручно лягають у руку і не виклкають болю.

– Слідкуй за своєю поставою. Сиди прямо і не нахиляйся до екрану.

– Захищай свій комп'ютер від вірусів, шпигунських програм і хакерів. Оновлюй ативірусне програмне забезпечення і не відкривай підозрілі файли і посилання.

– Не забувай про соціальну взаємодію. Спілкуйся з колегами, друзями і родиною. – Відвідуй заходи, які тебе цікавлять. Не ізолюй себе від світу.

– Люби свою професію, але не забувай про інші сфери життя. Розвивай свої захопленя, хоббі і таланти. Знаходь рівновагу між роботою і відпочинком.

Закон України “Про охорону праці” визначає основні принципи, завдання, права і обов’язки суб’єктів відносин з охорони праці, а також організаційні та правові основи державного управління і контролю за дотриманням законодавства про охорну праці.

Згідно з цим законом, ІТ компанії повинні впроваджувати такі заходи з охорни праці:

– Створювати на підприємстві службу охорони праці або призначати відповідальних осіб, які забезпечують розроблення, реалізацію та контроль за дотриманням заходів з охорони праці.

– Забезпечувати безпечні і нешкідливі умови праці для працівників, використвючи сучасні засоби техніки безпеки, санітарно-гігієнічні умови, засоби клектинго та індивідуального захисту, оптимальні режими праці та відпочинку.

– Проводити атестацію робочих місць на відповідність нормативно-правовим атам з охорони праці та аудит з охорони праці.

– Проводити навчання та інструктаж з питань охорони праці, з надання першої медичної допомоги потерпілим від нещасних випадків і правил поведінки у разі виникнення аварії[5].

– Забезпечувати лікувально-профілактичне обслуговування працюючих, санітарно-побутове обслуговування, пільги і компенсації для працівників, які працюють у важких і шкідливих умовах.

– Нести відповідальність за порушення законодавства про охорону праці та зподіяння шкоди життю і здоров'ю працівників.

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальна машин (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно створити належний мікроклімат, праметри якого регламентуються, Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98.

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

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

– ризик виникнення пожежі;

– негативний вплив на органи зору людини;

– ризики ураження електричним струмом;

– недостатня, або надмірна освітленість робочого місця;

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

– несприятливі мікрокліматичні умови;

– нервово-емоційна напруженість праці;

– інтелектуальні навантаження;

– монотонність праці;

– невідповідність ергономічних показників робочого місця діючим вимогам;

– шум;

– статичні навантаження на кістково-м’язовий апарат;

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

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

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	3
Довжина	4,6
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	6,9
Об'єм, V	$м^3$	не менше 20.0	20,7

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

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

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

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату

складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

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

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура, °С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-24	40-55	0,12
Тепла	23-25	50-70	0,1	24-25	50-65	0,9

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 Xerox WorkCentre 3025BI (3025VBI), електродвигуни вентиляторів ЕОМ.

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

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

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5-28:2018 [1], можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк. [1], Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

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

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

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значеннь;

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

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

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

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

8.5 Розрахункова частина

Початкові дані для розрахунку захисного штучного заземлення: опір заземлювача, який нормується: $R_{ЗН} = 4$ *Ом*.

Для захисного штучного заземлення застосовуються вертикальні електроди з металевого прутка діаметром 30 мм. (*D=30 мм.=0,03 м.)* довжиною *L*=2,5 *м.* та горизонтальний електрод – металева полоса з перетином 40*4 мм. тип грунта – глина (питомий опір 40 Ом*м). Відстань між вертикальними заземлювачами (електродами) *A*=3 *м*.

Глибина закладення горизонтального контура заземлення *t= 0,8 м*.

Умовна товщина верхнього шару грунта: *H=0,4 м*. Напруга – 220/380 В. Розрахункова схема розташування заземлюючих електродів – у ряд.

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача.

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

Розрахунок захисного заземлення можна автоматизувати за допомогою програми, сирцевий код якої опублікован на стр. 13-16 [6], або аналогічної.

Розрахунок

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,8+2,5/2=2,05 \text{ м.}$$

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

$$\rho = \psi \rho_2 = 1,36*40=54,5 \text{ Ом*м.}$$

де $\psi = 1,36$ – табличне значення коефіцієнта сезонності для відповідной климатичной зони у багатошаровому грунті [8];

$\rho_1 = 50$ *Ом*м.* – табличне значення питомого опору верхнього шару грунта [6];

$\rho_2 = 40$ *Ом*м.* – табличне значення питомого опору нижнього шару грунта [8].

Опір розтіканню електричного струму одного електрода вертикального заземлювача [8]:

$$R_o = 0,366 \frac{\rho}{L}\left(lg\frac{2L}{D}+\frac{1}{2}lg\frac{4T+L}{4T-L}\right)= 0,366\frac{54,5}{2,5}\left(lg\frac{2\cdot 2,5}{0,03}+\frac{1}{2}lg\frac{4\cdot 2,05+2,5}{4\cdot 2,05-2,5}\right)= 20,1 \text{ Ом.}$$

Відношення $A/L=3/2,5=1,2$.

Визначаємо коефіцієнт екранування ветикальньних електродів $Кев=0,8$ при попередній (орієтовній) кількості ветикальньних електродів, яке дорівнює 4 [8].

Визначаємо необхідну кількість вертикальних заземлювачів (без вхарування горизонтального заземлювача), при $R_{3H} = 4$ *Ом* :

$$N=R_O /(Кев\ R_{3H})= 20,1 / (0,8*4)=5,87 \approx 6 \text{ шт.}$$

Визначаємо довжину з'єднуючої полоси [8]:

$$L_П =1,05*A*N= 1,05*3*6=18,8\approx 19 \text{ м.}$$

Опір розтіканню електричного струму з’єднуючої полоси [8]:

$$R_П = 0{,}366(\rho_2 * K_П / L_П) lg(2(L_П * L_П)/(К * t)) =$$

$$= 0{,}366(40*5/16)*[lg(2*16*16)/(0{,}04*0{,}8)] = 16{,}5 \text{ Ом}.$$

де $K_П = 5$ – табличне значення коефіцієнта сезонності для відповідної климатичной зони з’єднуючої полоси: [8].

Загальний опір розтіканню електричного струму заземлювача [8]:

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

$$= (20{,}1 * 16{,}5)/(20{,}1 * 0{,}75 + 6 * 16{,}5 * 0{,}8) = 3{,}32 \text{ Ом}.$$

де $\eta_П = 0{,}75$ – табличне значення коефіцієнта екранування з’єднуючої полоси [8].

Умова $R \leq R_{ЗН}$ виконується ($3{,}32 \leq 4$).

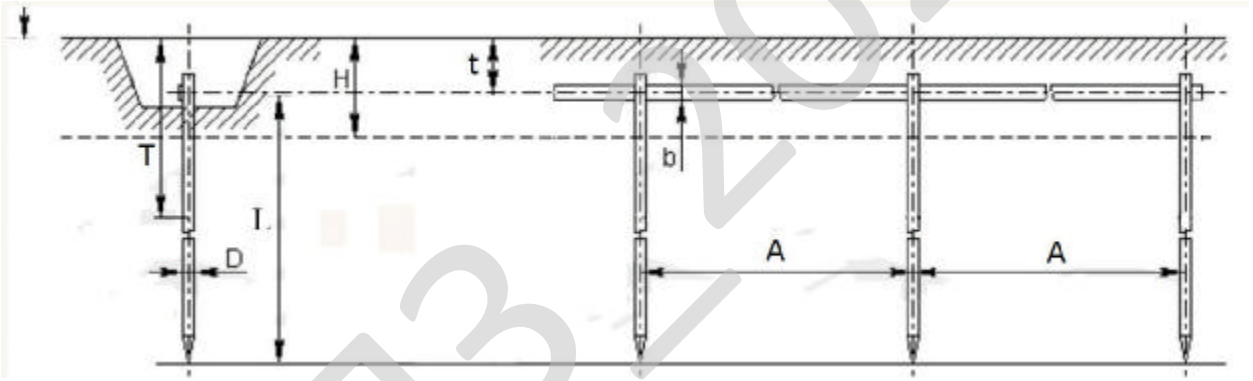

Рисунок 8.1 – Схема штучного заземлення

Висновки до розділу

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

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

Список використаних джерел інформації

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

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. 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

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

5. 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

6. 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

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

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

9. 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 19.09.22).

10. Методичні рекомендації до виконання розділу "Заходи з охорони праці та техніки безпеки" випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти для здобувачів вищої освіти спеціальностей 123 "Комп'ютерна інженерія" та 122 "Комп'ютерні науки" / М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т, каф. кібербезпеки та програм. забезпечення; [укл. О.В. Оришака, К.М. Марченко]. – Кропивницький: ЦНТУ, 2022. – 19 с. [Електронний ресурс]. – Режим доступу : http://dspace.kntu.kr.ua/jspui/handle/123456789/12240

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи диспетчеризації файлів у мобільній операційній системі iOS 16.

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 диспетчеризації файлів у мобільній операційній системі iOS 16.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем диспетчеризації файлів у мобільній операційній системі iOS 16.

– Досліджена система диспетчеризації файлів у мобільній операційній системі iOS 16.

– На основі отриманих результатів досліджень створена програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання диспетчеризації файлів у мобільній операційній системі iOS 16.

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 16.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ДСТУ 28147:2009.

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 6049 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,45 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пархоменко Д.О. Дослідження та програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16 // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2023.

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

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

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

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

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

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

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

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

10. 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.

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

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

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

14. 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.

15. 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.

16. 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.

17. 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.

18. 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

19. 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.

20. 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.

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

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

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

33. 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.

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. 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.

40. 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).

41. 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.

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

43. 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.

44. 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.

45. 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

46. 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.

47. 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.

48. 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.

49. 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.

50. 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.

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

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

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи диспетчеризації файлів у мобільній операційній системі iOS 16.

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. № 35-13 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 диспетчеризації файлів у мобільній операційній системі iOS 16.

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 диспетчеризації файлів у мобільній операційній системі iOS 16;

– 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 мобільному пристрої під керуванням ОС iOS 16 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС iOS 16.

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Objective-C.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Економічні вимоги

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

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

8 Вимоги щодо охорони праці

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

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 104 аркуші.

10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

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

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 13.12.2023 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

_____ Дрєєв О.М.

Дослідження та програмна реалізація системи диспетчеризації файлів у мобільній операційній системі iOS 16

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 44

Літера: РП

Кропивницький – 2023 року

AFCDevice.m – клас-оболонка для класу AFCInterface

```
	/*

	AFCDevice.m
	iPhoneConnection

	 Copyright (c) Пархоменко Дмитро Олексійович, 2023

	 */

// 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) Пархоменко Дмитро Олексійович, 2023

	 */

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

@interface AFCDevice : NSObject {

	AFCInterface *deviceInterface;
	AFCDeviceRef *device;

	id delegate;
}

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

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


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

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

-(AFCInterface *)deviceInterface;

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


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

@end
```

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

#ifndef MOBILEDEVICE_H
#define MOBILEDEVICE_H

#ifdef __cplusplus
extern "C" {
#endif

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

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

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

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

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

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

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

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

typedef unsigned int afc_error_t;
typedef unsigned int usbmux_error_t;

struct am_recovery_device;
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

typedef unsigned long long afc_file_ref;

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

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

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

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

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

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

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

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

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

CFMutableDictionaryRef AMRestoreCreateDefaultOptions(CFAllocatorRef allocator);

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

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

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

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

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

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

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

mach_error_t AMDShutdownNotificationProxy(void *);

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

#endif
```

Folder.m – робота з папками

```
      /*

     Folder.m
     iPhoneConnection

      Copyright (c) Пархоменко Дмитро Олексійович, 2023

      */

#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) Пархоменко Дмитро Олексійович, 2023

       */

#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) Пархоменко Дмитро Олексійович, 2023

    */

#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) Пархоменко Дмитро Олексійович, 2023


       */

#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) Пархоменко Дмитро Олексійович, 2023

  */

// 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) Пархоменко Дмитро Олексійович, 2023

      */

#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) Пархоменко Дмитро Олексійович, 2023
       */


#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) Пархоменко Дмитро Олексійович, 2023
       */

#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) Пархоменко Дмитро Олексійович, 2023
*/

#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) Пархоменко Дмитро Олексійович, 2023

       */

#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) Пархоменко Дмитро Олексійович, 2023

 */

#import <Cocoa/Cocoa.h>


@interface DisconnectedView : NSView {

}

@end
```

AFCFactory.m – підключення / відключення повідомлень

```
	/*

	AFCFactory.m
	iPhoneConnection

	 Copyright (c) Пархоменко Дмитро Олексійович, 2023
	 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) Пархоменко Дмитро Олексійович, 2023
       */

// 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) Пархоменко Дмитро Олексійович, 2023

	 */

#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
```