Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____комп'ютерних наук_____
(повна назва)

Кафедра _____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

ДОСЛІДЖЕННЯ МЕТОДІВ РОЗРОБКИ
ЕНЕРГОЗБЕРІГАЮЧОГО ПЗ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ
(тема)

Виконав: студент 2 курсу, групи ІПЗм-18-4__
спеціальності _121 – Інженерія програмного забезпечення_____
(код і повна назва спеціальності)
освітньо-наукової програми _Інженерія програмного_забезпечення_ _____
(повна назва освітньої програми)

_____Дригулич Є.С._____
(прізвище, ініціали)

Керівник _____проф. Шостак І.В._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук_____

Кафедра програмної інженерії_____

Рівень вищої освіти _____другий (магістерський)_____

Спеціальність _121 – Інженерія програмного забезпечення_____
(код і повна назва)

Освітньо-наукова програма Інженерія програмного забезпечення _____
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

Студентові _Дригуличу Євгену Сергійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Дослідження методів розробки енергозберігаючого ПЗ для мобільних пристроїв____

затверджена наказом по університету від «__»_____2020 р № ____

2. Термін подання студентом роботи до екзаменаційної комісії «11» травня 2020 р.

3. Вихідні дані до роботи *Алгоритми мобільних додатків та обробки мобільних даних, алгоритми захисту даних, методи стримінгу великих даних та пояснювальна записка. Використовувати ОС Windows, середовище об'єктно-орієнтованого проектування.*_____
_____
_____

4. Перелік питань, що потрібно опрацювати в роботі _*мета роботи, аналіз проблемної галузі і постановка задачі, методи пошуку корисних даних, опис об'єктних моделей, використовувані. методи та алгоритми, архітектура програмної системи, опис розробленої програмної системи. , результати тестування програмної системи*_____
_____

5. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	проф. Шостак І.В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз предметної галузі	25 березня 2020 р.	
2.	Огляд існуючих методів	31 березня 2020 р.	
3.	Методи створення та аналізу енергозберігаючого ПЗ	15 квітня 2020 р.	
4.	Підготовка пояснювальної записки	20 квітня 2020 р.	
5.	Спецчастина	28 квітня 2020 р.	
6.	Підготовка презентації та доповіді	03 травня 2020 р.	
7.	Попередній захист	05 травня 2020 р.	
8.	Нормоконтроль, рецензування	07 травня 2020 р.	
9.	Занесення роботи в електронний архів	08 травня 2020 р.	
10.	Допуск до захисту в зав. кафедри	10 травня 2020 р.	

Дата видачі завдання _ «____»_ _____2020 р.

Студент _____
(підпис)

Керівник роботи _____ __проф. Шостак І.В._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 85 с., 8 табл., 33 рис., 3 дод., 27 джерел.

ПРОДУКТИВНІСТЬ СКБД, ПОРІВНЯННЯ КЛАСТЕРНИХ ІНДЕКСІВ НА ПІДСТАВІ РІЗНИХ СТРУКРУР, ДЕРЕВА, ДЕРЕВА ВАН ЕМДЕ БОАСА

Мета дослідження – продовження тривалості роботи мобільного пристрою, без підзарядки від джерела живлення, за рахунок оптимізації алгоритмів роботи ПЗ, за критерієм мінімуму енергоспоживання.

Об’єкт дослідження – вплив алгоритмів і режимів роботи ПЗ на енергоспоживання мобільного пристрою.

Метод дослідження – моделі та методи розробки алгоритмів ПЗ з урахуванням енергоспоживання мобільного пристрою.

В результаті розроблено алгоритми оптимізації роботи ПЗ для мобільних пристроїв за критерієм мінімуму енергоспоживання.

MOBILE APPLICATIONS, GREEN SOFTWARE, ENERGY EFFICIENCY, ENERGY CONSUMPTION.

The purpose of research – extension of the time duration of mobile device work without recharging by optimization software algorithms by minimum power consumption.

Object of research – the impact of algorithms and modes of the software on the mobile device power consumption.

Subject of research – models and methods of development of software algorithms based on energy consumption of the mobile device.

Practical value – developed software optimization strategies can reduce the power consumption of the mobile device.

ЗМІСТ

ВСТУП

В епоху, коли швидко розвиваються інформаційні технології, постійно зростаючої кількості мобільних пристроїв і кількості ПЗ для них, особливо гостро стоїть проблема швидкого розряду акумуляторів. Дана проблема з'явилася не тільки з за межі технологій створення накопичувальних осередків акумуляторів, а й через неправильного підходу до створення ПЗ під мобільні платформи [1].

Велика кількість оперативної пам'яті, висока частота процесорів і т.д. відучили розробника економити ресурси пристрою, оптимізуючи свій код, тим самим прискорюючи розробку ПЗ. Але для оптимізації по енергоспоживанню все ж треба вносити ряд поліпшень в логіку роботи ПЗ, наприклад: можна зменшувати частоту відтворення графіки в іграх при зміні джерела живлення з мережевого на акумуляторне, робити обчислення в декількох потоках використовуючи процесор на максимум, а так само виробляючи запис на диск пакетами (збираючи дані в великі пакети і лише потім записуючи).

Зараз набирають популярність ряд методик, які можна об'єднати під однією назвою – «зелені додатки» (Green software) [2] і більшість популярних операційних систем вже успішно застосовують їх, що не заважає, впровадять ті ж підходи до розробки окремих випадків мобільного ПЗ . Однак у зв'язку з новизною і малої вивченістю цієї теми більшість розробників ПЗ, все ще не застосовують дані рішення в комерційній розробці.

Мета атестаційної роботи – розробка алгоритмів продовження тривалості роботи мобільного пристрою, без підзарядки від джерела живлення, за рахунок оптимізації алгоритмів роботи ПЗ, за критерієм мінімуму енергоспоживання.

1 АНАЛІЗ РОЗРОБКИ ЕКОЛОГІЧНИХ ПЗ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

1.1 Аналіз методів енергоефективності мобільних пристроїв

Багато з сучасних мобільних пристроїв не поступаються за потужністю стаціонарним ПК і одночасно з цим володіють такими властивостями, як портативність і переносимість. Але ці плюси так само тягнуть за собою ряд, досить таки значних проблем: проблема з виведенням надлишків тепла утвореного при роботі таких компонентів як процесор, пам'ять і т.ін. І проблему малого терміну роботи пристрою без підключення до мережі електроживлення. Обидві ці проблеми мають два шляхи вирішення, які не є взаємовиключними – це застосування нових технологій для створення даного компонента (процесор, пам'ять , акумулятор) І впровадження нових підходів до розробки ПЗ для даних компонентів.

Можна використовувати тільки один з цих підходів, але тим самим обмежуючи себе, так як, переносячи старі рішення (підходи) на нові компоненти розробник занижує їх (компонентів) можливості і збільшуємо технічний борг.

Докладно розглянуто проблему швидкого розряду акумулятора при роботі без підключення до мережі електроживлення. Крім незручностей у використанні, швидкий розряд несе за собою одну проблему – утилізацію елементів живлення після закінчення термінів їх експлуатації. Зараз використовуються такі види акумуляторів: свинцево-кислотні та срібно-цинкові.

Свинцево-кислотні акумулятори застосовуються в автомобілях, електромобілях, мотоциклах, джерелах безперебійного живлення, різному промисловому устаткуванні [3]. Свинець є токсичним металом і, потрапляючи в організм, він накопичується в кістках, викликаючи їх руйнування. Кислоти, зокрема найбільш поширена у виробництві батарей сірчана, також досить небезпечні. При переробці подібних батарей спочатку нейтралізується кислота,потім корпус відділяється від свинцевих пластин, і все це використовується в переробці, в тому числі і для виробництва нових батарей.

Срібно-цинкові акумулятори найчастіше використовуються в наручних годинниках, дитячих іграшках, медичних пристроях і іншої малогабаритної техніки. Срібно-цинкові акумулятори містять вкрай шкідливу для навколишнього середовища і здоров'я людини ртуть,яка з часом починає роз'їдати стінки батареї і протікати, тому їх слід утилізувати з особливою ретельністю [4].

Таким чином, збільшуючи час розряду батареї мобільного пристрою, збільшується термін життя акумулятора, що в свою чергу знижує загальні витрати на утилізацію елементів живлення в цілому, також зменшуючи сумарне енергоспоживання мобільних пристроїв.

Технології не стоять на місці, зараз з'являється багато нових видів акумуляторних осередків, які можуть кардинально змінити час роботи мобільного пристрою без підключення його до мережі електроживлення.

Так само набирає обертів новий підхід, який шляхом незначної оптимізації логічних процесів, що відбуваються при виконанні ПЗ, відчутно збільшить час роботи акумулятора, а так же, з причини своєї специфіки, знизить час завантаженості пам'яті і процесора, що в свою чергу призведе до зменшення нагрівання даних компонентів .

Даний підхід відповідає таким принципам:

- використовувати ресурси пристрою настільки, наскільки потрібно (немає сенсу проводити розрахунки в одному потоці, якщо пристрій має, наприклад, 4-х ядерним процесором);
- діяти в рамках контексту (працювати в різних режимах в залежності від того, що зараз є джерелом живлення: мережа або акумулятор).

Комбінація двох цих методів (поліпшення апаратної складової і оптимізація ПЗ) дозволить досягти небувалого терміну роботи пристрою без доступу до електромережі [4].

1.2 Аналіз програмних засобів мобільних пристроїв

Мобільні операційні системи, а саме iOS і Android швидко набирають популярність, про що свідчать останні дослідження ринку мобільних апаратів.

Темпи зростання продажів комп'ютерів (настільних і портативних) перестав збільшуватися і тримається на одному рівні вже приблизно два роки. Мобільні пристрої в свою чергу показують значне зростання продажів, до цих пристроїв відносяться і планшетні комп'ютери [5].

Нижче ви можете побачити графік зміни продажів мобільних пристроїв, комп'ютерів за всю історію. Так само на графіку представлені зміни продажів різних пристроїв, не тільки PC або телефони. За допомогою них можна наочніше оцінити ситуацію (рисунок 1.1).

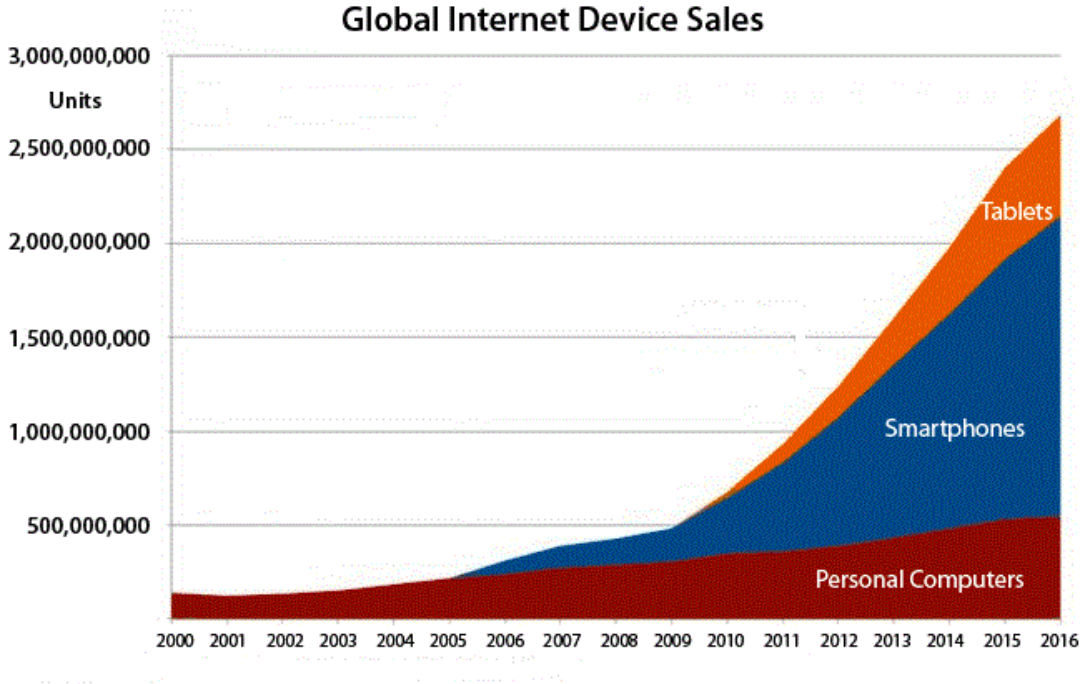

Рисунок 1.1 – Графік популярності мобільних пристроїв [5]

Аналітична агенція International Data Corporation (IDC) опублікувало результати дослідження, присвяченого вивченню популярності мобільних програмних платформ за підсумками першого кварталу 2019 р. (рисунок 1.2).

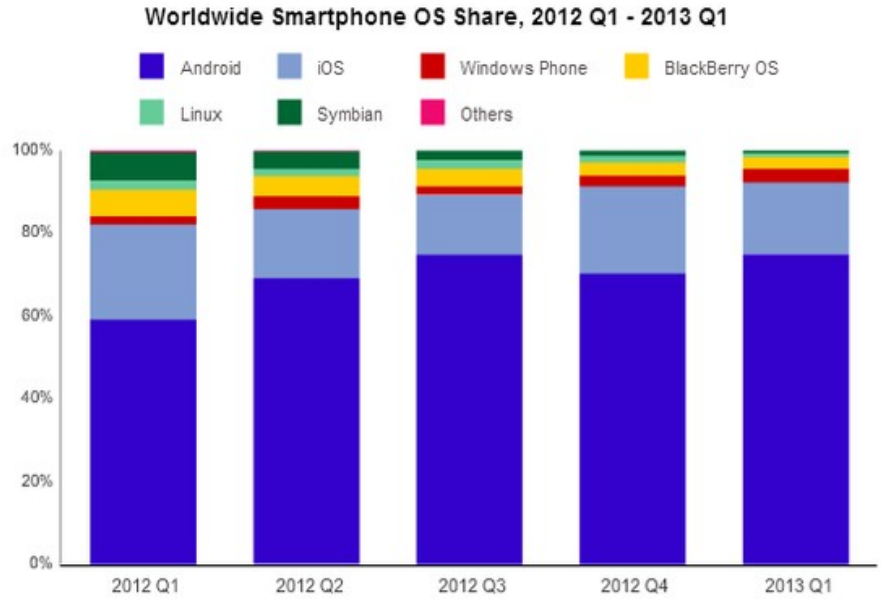

Рисунок 1.2 – Частка ринку різних мобільних пристроїв

Найпоширенішою операційною системою для смартфонів виявилася Google Android, близько 75% ринку, Apple iOS – 17,3%, Windows Phone – 3,2%, BlackBerry – 2,9%, Linux – 1%. У порівнянні з минулим роком позитивну динаміку зростання показують тільки перші три операційні системи [5].

1.3 Аналіз алгоритмів підвищення ефективності роботи ПЗ

На поточний момент більшість розробників ПЗ не приділяють увагу оптимізації. До таких висновків можна прийти, скориставшись будь-яким стандартним аналізатором під будь-якою платформою (memory monitor в Windows, Profiler в iOS.).

Це відбувається з невірного тлумачення терміна «період активності». Визначення «періоду активності» додатка звучить так: це час, протягом якого програма виконує корисну роботу.

Термін «Корисна робота» – це робота, необхідна для виконання дії, безпосередньо запитаного користувачем. Робота, яка виконується в фоновому режимі, не вважається «корисною роботою». Можна навести такі приклади періодів активності додатків: обчислення результатів в електронній таблиці або робота пошукової системи по підбору результатів, відповідних пошуковому запиту користувача. В обох випадках дії спочатку запитуються користувачем, але потім програма виконує роботу без втручання користувача. Серед інших прикладів – потокова передача відео або запуск перевірки на віруси [4].

З вище сказаного, очевидно, що доцільно економити заряд акумуляторів в періоди простою додатків, але, найчастіше упускається з виду можливість економії електрики і в ті періоди, коли виконується корисна робота. Якщо оптимізувати додаток так, що під час активної роботи воно буде споживати менше енергії, то і додаток стане працювати швидше, і користувачі будуть більш задоволені (за рахунок можливості довше працювати з пристроєм без підзарядки) і будуть більш охоче використовувати цю програму. Таким чином, це дуже вигідний для всіх результат.

У сучасних пристроях все частіше використовуються багатоядерні процесори. Використання всіх доступних ядер сприяє не тільки підвищенню продуктивності, але і економії електроенергії. Чи має сенс навантажувати одне ядро протягом тривалого періоду на повну або майже повну потужність, якщо можна виконати завдання швидше (і економніше), використовуючи всі доступні ядра. Якщо домогтися одночасного використання найбільшого числа потоків або ядер, то програма буде працювати швидше, витрачаючи менше електроенергії.

На наведеному нижче графіку (рисунок 1.3) показано споживання електрики трьома різними додатками при тестуванні на процесорі Intel® Core™ другого покоління в операційній системі Windows® 7 [1]. Найбільшу перевагу багатопотокового режиму демонструє мультимедіа-додаток, хоча всі три програми витрачають менше електрики в інтер режимі.

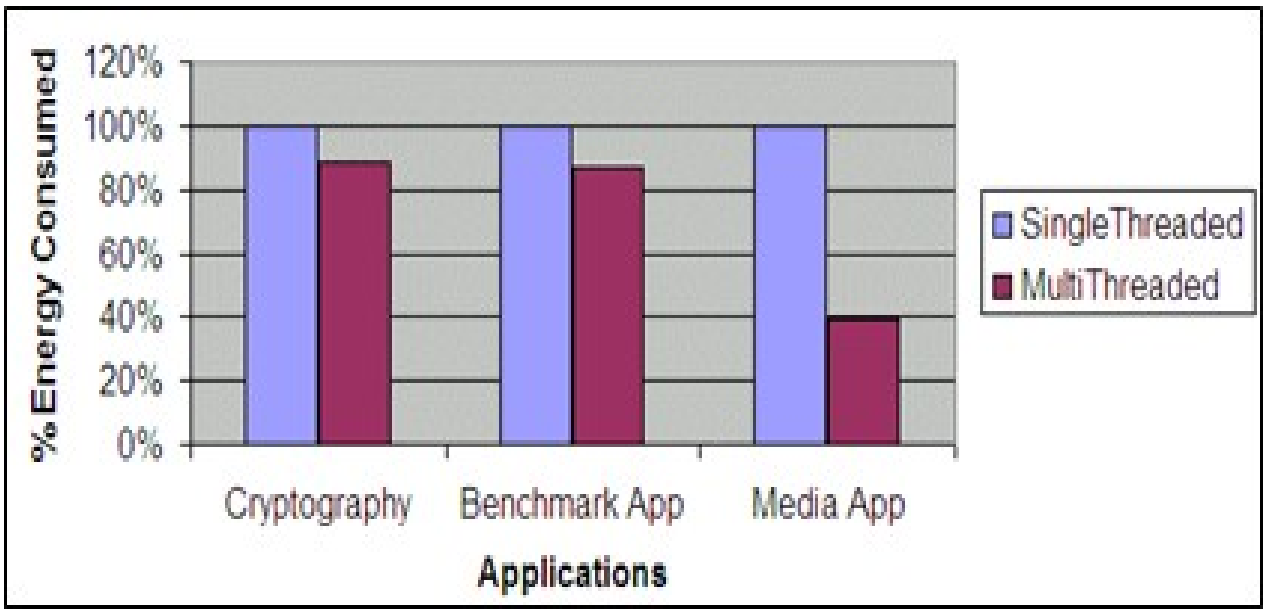

Рисунок 1.3 – Економія електроенергії трьох різних додатків в інтер режимі

Однією з основних причин високого споживання електрики в активному стані програми є висока частота системних викликів. У додатку може виникнути конфлікт між потоками, що взаємодіють з ядром системи. Щоб уникнути цих проблем потрібно використовувати виклик API Windows «EnterCriticalSection ()» для синхронізації передачі даних між потоками в просторі користувача, а не виклик API Windows «WaitForSingleObject ()», що виконується в просторі ядра. Економія споживання електроенергії 4-х потоковим додатком при зниженні числа блокувань через конфлікти може досягти 60% (за результатами тестування на процесорі Intel Core™ 2-го покоління на платформі Windows 7) [1] Графік аналізу між WaitForSingleObject () і EnterCriticalSection () в 4-х потоковому додатку. Дані отримані для процесора Intel® Core™ 2-го покоління на платформі Windows 7 (рисунок 1.4).

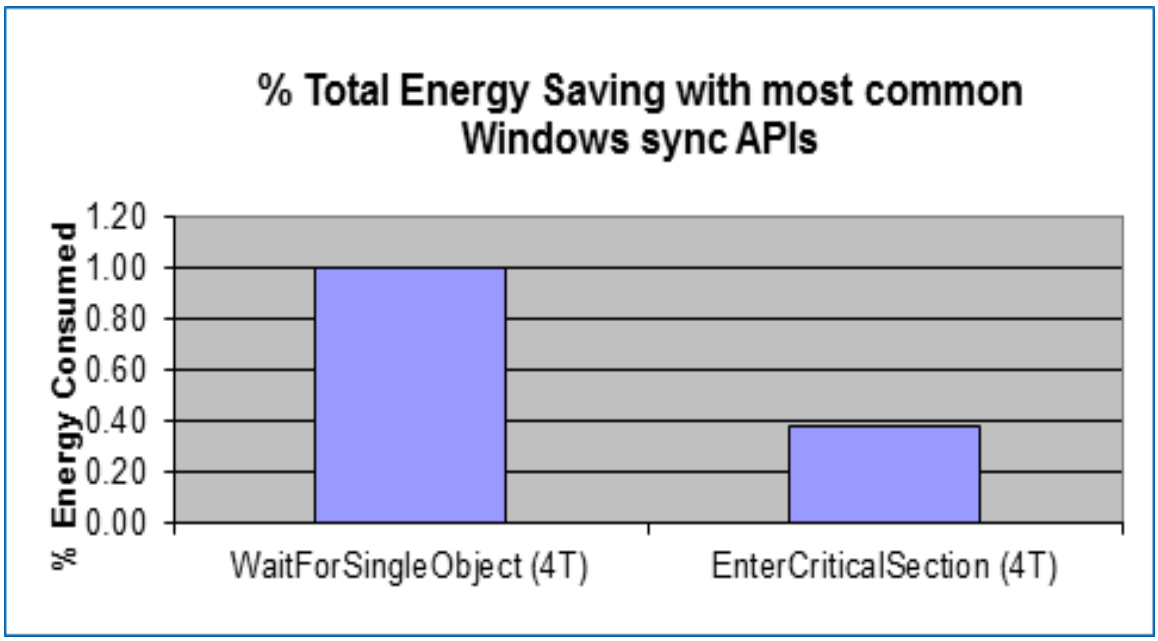

Рисунок 1.4 – Порівняння показників економії енергії

Важливим джерелом викликів синхронізації на рівні ядра є переходи між активним станом і станом простою. Якщо об'єднати періодичні дії в групи і уникнути зайві переходи між активним станом і станом простою, то вдасться підвищити паралельність потоків. Ці зміни, доповнені багато поточностью, приведуть до зниження споживання електроенергії додатком [1].

Якщо використовувати тривалу періодичність системного таймера і уникати частих періодичних опитувань, можна знизити кількість енергії, витраченої на очікування ресурсів системи [3].

Можна знизити споживання живлення, якщо об'єднати сеанси введення-виведення дисків в пакети. Як показано в таблиці нижче, можна домогтися економії електроенергії за рахунок продовження періодів простою дисків між сеансами доступу.

Цей принцип стосується і будь-яких дій, що призводять до зміни стану додатків. Якщо об'єднати виклики ресурсів в пакети, додаток буде споживати менше електрики (рисунок 1.5).

Access Interval	100 mS	1S	5S	10 S	20S
SSD Average Power	0.5W	208mW	154mW	89mW*	
HDD Average Power	1.4 W	1.1 W	1.02 W	1.14 W	0.98 W

Рисунок 1.5 – Приклад оптимізації витрати електрики системою зберігання даних

Оптимізація використання оперативної пам’яті також допоможе знизити споживання електроенергії додатком в активному стані.

Ось кілька рекомендацій:

- уникати непотрібного перетворення формату графіки (наприклад, перетворення з формату YUV в RGB і назад) для мінімізації числа звернень до графічного і центральному процесорам;
- застосовувати кеш для часто використовуваних структур даних;
- обмежувати кількість переміщень даних між простором ядра і простором користувача [3].

Деякого виграшу можна досягти, використовуючи більш економічні (з точки зору витрати електрики) методи. У наведеній нижче таблиці показано кілька методів Microsoft DirectX Present і відповідні показники витрати електрики при відтворенні відео. Ці показники отримані для процесора Intel Core™ 2-го покоління на платформі Windows 7.

У цьому прикладі кожен кадр (відео високої чіткості) займає в пам’яті або 2 байта, або 4 байти (при використанні формату, відповідно, YUY2 або RGB). Отже, формат YUY2 використовує менше пам’яті і є більш ефективним [4] (рисунок 1.6).

DX9 Present Method	Memory BW (GB/s)	Package Power (W)	Memory Power (W)
CopyBlt	2.57	2.88	1.02
RGB Overlay	1.37	2.62	0.76
YUY2 Overlay	0.93	2.48	0.63

Рисунок 1.6 – Витрата електрики методами DX9 Present
для процесора Intel Core™ 2-го покоління на платформі Windows 7

Намагаючись знизити витрату електроенергії в активному стані додатків, пам’ятайте, що продовжити термін роботи пристроїв від акумуляторів можуть і додатки, що змінюють режим своєї роботи в залежності від використовуваного в даний момент джерела електроенергії (мережа або акумулятор) і від заряду акумулятора [5].

1.4 Аналіз застосування комбінованих стратегій

У даній області найкращих результатів можна досягти, використовуючи комбінування існуючих підходів. Кожне ПЗ по-своєму унікальне, але можна виробити загальні стратегії, які будуть прийнятні для більшості завдань.

Нижче наведені приклади оптимізації:

- 3D-ігри – коли система оповіщає про роботу від акумуляторів, можна обмежити максимальну швидкість кадрів більш низьким значенням;

- відеопрогравачи – при низькому рівні заряду акумулятора можна менше використовувати або відключити фільтри поліпшення чіткості і кольорів зображення при відтворенні;

- не потрібно оновлювати екран, коли додаток згорнуто;

- виберіть параметри за замовчуванням, оптимальні з точки зору економії електрики;

- при роботі від акумулятора слід відкладати другорядні завдання;

- пропонуйте користувачам додаткові способи скорочення витрати електрики у вигляді необов'язкових параметрів.

Окрему увагу варто приділити смартфонам і планшетів. Якщо ноутбук може тривалий час бути постійно підключеним до мережі і, по суті, бути стаціонарним комп'ютером, то смартфон в такому режимі для нас буде марний. Час функціональної життя смартфона набагато нижче, ніж ноутбука або подібної йому техніці. Смартфони набагато швидше морально застарівають і заміняються на більш нові моделі, їх акумулятори частіше проходять повний цикл розрядки. Саме їх недовгим періодом функціонування без підзарядки користувач незадоволений найбільше. Все вищесказане повною мірою відноситься і до планшетів [6].

З огляду на те, що акумулятори смартфонів схильні до більшого стресу, вони найчастіше і приходять в непридатність, перестають утримувати заряд і, як наслідок, замінюються і утилізуються.

На жаль, на багатьох рясних платформах доступ до режимів роботи процесора або ще якогось внутрішнього компонента закритий і тут шляхів для зниження набагато менше. Розглянемо найбільш поширені мобільні платформи (iOS, Android, Windows Phone, BlackBerry OS) [7].

Android (Андроїд) – портативна (мережева) операційна система для комунікаторів, планшетних комп'ютерів, електронних книжок, цифрових

програвачів, наручних годинників, нетбуків і смартбуков, заснована на ядрі Linux. Спочатку розроблялася компанією Android Inc., яку потім купила копання Google. Згодом Google ініціювала створення альянсу Open Handset Alliance (OHA), який зараз і займається підтримкою і подальшим розвитком платформи. Android дозволяє створювати Java-додатка, що керують пристроєм через розроблені Google бібліотеки. Android Native Development Kit створює додатки, написані на Сі та інших мовах [8].

iOS (до 24 червня 2010 року – iPhone OS) – мобільна операційна система, що розробляється і випускається американською компанією Apple. Була випущена в 2007 році, спочатку – для iPhone і iPod touch, пізніше – для таких пристроїв, як iPad і Apple TV. На відміну від Windows Phone і Google Android, випускається тільки для пристроїв, вироблених фірмою Apple.

Інтерфейс iOS заснований на концепції прямого маніпулювання з використанням жестів мультитач. Елементи управління інтерфейсом складаються з повзунків, перемикачів і кнопок.

iOS розроблена на основі Mac OS X і використовує той же POSIX-сумісний набір основних компонентів Darwin. В iOS є чотири шари абстрагування: шар Core OS, шар Core Services, шар Media Layer, і шар Cocoa Touch. Для поточної версії операційної системи iOS виділяється 1,5 – 2 Гб флеш-пам'яті пристрою для системного розділу і приблизно 800 Мб вільного місця (варіюється в залежності від моделі).

Плюсами Android є повна відкритість, можливість отримати доступ до системних функцій, а також можливість створювати призначені для користувача версії даної ОС. Мінусами є наявність шкідливих програм (архітектура дозволяє писати _ористувача ін програми для інфікування інших пристроїв), не жорсткий контроль додатків, які розповсюджуються через стандартний Android магазин.

На відміну від Android, iOS є прикладом закритою мобільної платформи, доступ до системних функцій умовно обмежений (специфіка мови Objective C не допускає прихованих методів і реалізацій). Плюсами iOS є: якість додатків, які розповсюджуються через стандартний магазин і відсутністю шкідливих ПЗ. Мінус

iOS – закритість самої платформи. Деякі речі не можливо реалізувати чисто фізично, внаслідок відсутності доступу до внутрішніх API [10].

Практично всі методи, що описані, застосовні в Android в слідстві відкритості платформи, але в iOS можна використовувати тільки деякі з них.

Незважаючи на відмінності iOS і Android для них можна підібрати одну загальну стратегію, яка хоч і не вирішить всіх проблем, але дозволить знизити енергоспоживання до мінімуму. Кроки щодо зниження енергоспоживання мобільного ПЗ (Android, iOS):

- додаток не повинно працювати в фоні (додаток в згорнутому стані повинна зберегти свої дані і підготується до видалення з пам’яті);
- додатки, які знаходяться в згорнутому стані, через деякий час повинні побиватися з оперативної пам’яті (в iOS цей час для звичайних додатків становить 10 хвилин);
- додаток повинен оптимально використовувати ресурси пристрою (наприклад, не має сенсу постійно оновлювати дані, краще використовувати нотифікації, які будуть надсилатися тільки тоді, коли дані змінилися) [11].

Таким чином, використовуючи комбіновані стратегії на мобільних пристроях, таких як смартфони та планшети, можна добитися значного зниження енергоспоживання при мінімальних витратах на оптимізацію ПЗ.

Зниження витрати енергоспоживання мобільного пристрою призводить до збільшення терміну служби і часу розряду акумулятора. Навіть сама мінімальна оптимізація додатки, яке користується популярністю у користувача, веде до відчутного зниження енергоспоживання. В іншому випадку, навіть одна програма в тлі, що не економно ставиться до ресурсів, значно впливає на енергоспоживання самої системи.

Дана тема досить таки нова, але вже зараз видно її перспективи. Для демонстрації, наскільки ПЗ впливає на енергоспоживання пристрою, варто навести приклад Android смартфона і ряду мінімальних дій, які покажуть, що за

своїм енергоспоживанню даний пристрій ніяк не відрізняється від інших смартфонів, хоча і є думка, що Android найменш економно ставиться до ресурсів батареї.

Нижче наведені кроки по мінімізації енергоспоживання для Android пристроїв:

- відключити всі сервіси синхронізації;
- відключити фонову передачу даних;
- відключити всі програми, які можуть в тлі завантажувати дані;
- уникати будь-яких програм, що працюють у фоновому режимі;
- відключити мобільний інтернет;
- перевести телефон з 3G в режим EDGE.

Тепер пристрій може пропрацювати автономно до 40 годин, правда при активному використанні всіх відключених функцій цей час може скоротитися до 3 – 5 годин.

Дана тема зачіпає найвразливіше місце сучасних мобільних технологій – енергоспоживання, і пропонує ряд дешевих у впровадженні рішень щодо зниження енергоспоживання пристрою. Також дана тема розкриває і показує відповідальність розробника ПЗ за малий термін автономної роботи пристрою і вирішення даної проблеми є технічно боргом розробника [13].

1.5 Мета і постановка задач дослідження

Існує цілий комплекс заходів щодо зниження енергоспоживання мобільного пристрою за рахунок модернізації ПЗ, частина з них використовується вже зараз, реалізація ж інших дозволить значно знизити енергоспоживання мобільного пристрою. Також варто відзначити, що мобільні платформи для планшетів і смартфонів накладають ряд обмежень по можливості оптимізації енергоспоживання за рахунок закритості системи, але все одно для них теж можна створити стратегії по мінімізації енергоспоживання.

При аналізі способів зниження енергоспоживання мобільного пристрою за рахунок оптимізації ПЗ, доведена необхідність впровадження стратегій оптимізації мобільного ПЗ. Так само розглянуті види мобільних платформ і особливості оптимізації ПЗ для планшетів і смартфонів. Проведений аналіз існуючих способів зниження енергоспоживання за рахунок підвищення ефективності ПЗ і прикладів стратегій, довів необхідність застосування комплексних підходів щодо оптимізації енергоефективності ПЗ мобільних пристроїв, тим самим підвищуючи свою ефективність.

Мета дослідження – продовження тривалості роботи мобільного пристрою, без підзарядки від джерела живлення, за рахунок оптимізації алгоритмів роботи ПЗ, за критерієм мінімуму енергоспоживання.

Завдання:

- виконати аналіз проблеми енергоспоживання мобільних пристроїв при існуючих підходах до розробки ПЗ;
- виконати аналіз можливості впровадження нових підходів до розробки енергоефективного додатку;
- розробити загальні стратегії для різних типів мобільних пристроїв і платформ;
- сформулювати вимоги і вихідні дані для прототипу екологічно чистого додатку;

- розробити прототип додатка для моделювання розроблених алгоритмів;
- виконати тестування прототипу ПЗ;
- оптимізувати алгоритми роботи програми прототипу;
- виконати порівняльну оцінку внесених оптимізацій і визначити їх доцільність.

2 ОПИС МОДЕЛЮВАННЯ МЕТОДІВ РОЗРОБКИ ЕКОЛОГІЧНОГО ПЗ

2.1 Методи розробки енергозберігаючого програмного забезпечення

Протягом багатьох років виробники мобільних платформ шукали способи продовження часу роботи батарей мобільних пристроїв. Згодом були значно вдосконалені технології виробництва акумуляторів, почали застосовуватися нові, менш енерговитратні, режими роботи процесора, було істотно оптимізовано енергоспоживання дисплеїв. Однак можливості для подальшого розвитку ще аж ніяк не вичерпані. Важливу роль в зниженні кількості споживаної енергії на мобільних платформах і продовження терміну служби батарей грає програмне забезпечення [12].

Режими P і C енергоспоживання процесора. Процесор являє основний інтерес для розробників програмного забезпечення. Розуміння певних станів енергоспоживання процесора може допомогти в боротьбі за енергоефективність. Важливо, щоб в періоди, коли процесор зайнятий активною обробкою інформації або виконанням обчислень, він споживав мінімальну кількість енергії. Процесор має С-стану і Р-стану. С-стану – це режими харчування ядра, які визначають ступінь «сну» процесора. У стані С0 процесор активний і виконує певні команди. У цьому режимі, процесор може працювати на різних рівнях частоти, в залежності від Р-стани [12].

Між періодами активності процесор має можливість для відпочинку. Насправді, С-стану часто означають стану «сну». Процесори Intel® підтримують кілька рівнів С-станів ядра і пакетів (ресурси, загальні для всіх ядер), які забезпечують гнучкий вибір між споживанням енергії та чуйністю. З кожним глибшим рівнем сну, деякі нові частини процесора вимикаються, в результаті чого зберігається більше енергії. Чим глибше сон, тим більше енергії економиться. Навіть якщо період сну становить всього 100 мікросекунд, з часом значна кількість енергії може бути збережено.

P-стани, або стани виконання, визначають частоту, з якою процесор працює. Різні марки процесорів представляють P-стани як функції, такі як SpeedStep в процесорах Intel, PowerNow !, Cool'n'Quiet в процесорах AMD і PowerSaver в процесорах VIA. Стандартні P-стани:

- P0 – максимальна потужність і частота;
- P1 – менше, ніж в P0, напруга і частота диференційовані;
- Pn – низька номінальна напруга і частота [12].

Після того, як було коротко розказано про енергоефективність компонентів і станів процесора, не можна не пояснити, чому дані аспекти важливі для розробника. Так як більшість енергозберігаючих функцій в платформі відомі розробникам програмного забезпечення і існує дуже мало способів прямого контролю з боку програми, поведінка програмного забезпечення робить істотний вплив на ефективність енергозберігаючих функцій, вбудованих в платформу. Добре розроблене програмне забезпечення дозволяє добре працювати механізмам енергозбереження. Погано розроблене програмне забезпечення, навпаки, пригнічує енергозберігаючі функції і призводить до зниження часу роботи батареї і зростанню витрат енергії.

Перш ніж перейти до опису методів розробки енергоефективних програм, важливо зрозуміти відмінність між активними і фоновими програмами. Активним називається програмне забезпечення, яке працює за своїм прямим призначенням, наприклад здійснює обчислення таблиці, відтворення музики або фільму, завантаження фотографій на веб-сайт, доступ в Інтернет і т. Д. У всіх цих випадках присутні навантаження, тому CPU і GPU зайняті роботою. Фонове програмне забезпечення – це програмне забезпечення, яке, по суті, працює, але очікує події, при настанні якого воно стане активним. Прикладами фонового програмного забезпечення є: запущений браузер, для якого не зазначений веб-сайт для перегляду; відкритий текстовий процесор, що знаходиться в фоновому режимі або вікно якого зменшено; програма обміну миттєвими повідомленнями, яка працює, але не відправляє і не приймає повідомлення в поточний момент.

2.2 Основні вимоги до оптимізації енергоспоживання додатків

Будь-яка оптимізація корисна, якщо витрати на неї окупаються і якщо це доцільно.

Розглянуто деякі з видів додатків і способи оптимізації їх за енерговитратами (оптимізація по енергоефективності часто покращує продуктивність, що в свою чергу позначається на загальній задоволеності користувачів).

Додаток, що виконує конвертацію або аналіз великих масивів даних. В першу чергу слід винести обчислення в фоновий потік, тим самим позбувшись «заморозки» UI під час проведення обчислень. Але цього не достатньо, так як всі обчислення проводяться лише в одному потоці, а ядер на мобільних пристроях часто більше одного або двох. Додатки подібного плану з за тривалості розрахунків часто згортають в фон, значить додаток крім усього іншого має оптимально працювати у фоновому режимі, а це значить, що розрахунки повинні бути проведені швидко, підготувавши додаток до закриття (більшість мобільних платформ стежить за тим, що б додаток побивалося з пам’яті після деякого часу перебування в тлі, в iOS це 10 хв). Так само має сенс зберегти всі дані після проведення розрахунків з метою відновити роботу після повторного включення програми та вивантажити їх з пам’яті, якщо невідомо коли вони можуть знадобитися [14].

Додаток програвач або інший фоновий – відмовитися від роботи в тлі не можна, але можна перевіряти активність користувача і в залежності від неї можна припиняти відтворення. Так само можна видати запит на дозвіл зниження яскравості екрану або його повне відключення. Не менш важливо спостереження за станом акумулятора, ці дані можуть дати нам інформацію про те, коли слід припинити відтворення та видати попередження користувачеві про негативну тенденцію в витраті заряду батареї [14].

Поштові клієнти та додатки, що вимагають синхронізації з інтернетом. Дані програми характеризуються тим, що з певною періодичністю звіряють свої статки зі станом даних на сервері. Залежно від характеру додатки рекомендується відмовитися від ітеративних оновлень за таймером на користь стандартних (системних) центрів нотифікацій. Якщо відмовитися від постійних оновлень за таймером можна, коштувати почати регулювати їх частоту в залежності від стану системи: рідкісні оновлення, коли додаток йде в фон, мінімізація або навіть відключення оновлень, коли заряд батареї наближається до критичного тощо [14].

Ігрові програми вимагають великої кількості ресурсів системи для своєї роботи і як наслідок цей тип додатків є великою область для оптимізації.

Найбільш очевидними видами оптимізації є виключення відтворення фонової музики, при згортанні додатка, а так само відключення відтворення. По можливості варто поставити гру на паузу і зупинити всі інтер процеси, але тут все залежить від характеру процесів і жанру гри. Повідомлення про зменшення запасу батареї і запиту на зменшення яскравості, не обтяжить користувача, а дасть йому можливість самому вирішувати, що для нього зараз пріоритетне. З більш радикальних оптимізацій є зниження якості ігрових текстур, частоти перемальовування та ін. Але такі дії варто робити тільки після запиту на дозвіл у користувача [14].

Розглянувши деякі види додатків і оптимізації під них, варто зауважити, що розробник додатків під мобільні платформи повинен враховувати всю важливість зниження енергоспоживання системою.

3 МОДЕЛЮВАННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ЕНЕРГОСПОЖИВАННЯ ПРИ РОЗРОБЦІ ПЗ

3.1 Формування таблиці критеріїв оцінки стратегій оптимізації ПЗ

Провівши загальний аналіз проблеми та склавши список можливих оптимізацій для роботи ПЗ, можна приступати до розробки плану дослідження.

Основні принципи оптимізації по енергоспоживанню це:

- вирішувати швидко;
- використовувати ресурси пристрою настільки, наскільки потрібно (немає сенсу проводити розрахунки в одному потоці, якщо пристрій має до прикладу 4-х ядерним процесором);
- діяти в рамках контексту (працювати в різних режимах в залежності від того, що зараз є джерелом живлення: мережа або акумулятор).

Метою даного дослідження є вироблення стратегій по економії заряду акумулятора для різних типів додатків. Одним з основних вимог до даних стратегій буде вважатися – простота впровадження, так як ніякої керівник проекту не захоче додавати в свій проект необов’язкову функціональність, якщо час на її реалізацію перевищить тиждень [14].

Для дослідження будуть написані або взяті open source програми і проведений аналіз енергоспоживання цих додатків під час роботи. Потім будуть складені стратегії і обрані найоптимальніші методи зниження витрати заряду акумулятора, за такими критеріями:

- результат (процентна характеристика відображає зменшення енергоспоживання);
- простота впровадження (умовна характеристика відображає швидкість внесення змін до проекту);

- вплив на загальне враження про програму (якщо для даної оптимізації потрібно постійно вимкнений дисплей – цим додатком ніхто не буде користуватися);

- вплив на якість і загальну швидкість роботи програми (якщо при впровадженні оптимізації постраждає якийсь інший компонент або загальна швидкість роботи, можливо варто відмовитися від даної оптимізації).

Результатом даного дослідження буде опис алгоритмів оптимізації, з детальним роз'ясненням всіх плюсів і мінусів, а так само простий покроковий підручник, який описує принципи інтеграції даного рішення в проект.

Розробка програмного забезпечення спрямована на створення кінцевого продукту і підтримку його працездатності. Сучасні реалії такі, що набагато важливіше створити, швидко, продукт задовільної якості, ніж ідеальний, але за більш тривалий період часу. Це змушує йти розробника на компроміси, жертвуючи якістю коду, його продуктивністю або безотказностью. Ті ж самі принципи можна застосувати і до оцінки рентабельності застосування тієї чи іншої стратегії оптимізації програмного забезпечення по енергоспоживанню. Наприклад: чи не доцільно застосовувати стратегію оптимізації, якщо за нею піде переробка інфраструктури всього проекту.

В даному підрозділі буде проведена оцінка основних критеріїв якими керується розробник при ухваленні рішення про внесення змін до початкового коду програмного продукту. Так само будуть присвоєні суб'єктивні (відштовхуючись від особистого досвіду комерційної розробки програмного забезпечення) вагові коефіцієнти кожного з результатів застосування стратегій і дані пріоритети кожному з використовуваних критеріїв згідно їх важливості.

Нижче перераховані критерії та обмеження, які вважаються основними і визначальними при ухваленні рішення на користь тієї чи іншої стратегії оптимізації програмного забезпечення по енергоспоживанню.

Обмеження трудовитрат вимірюється в людино-годинах. У поточному контексті є синонімом вартості зміни, так як, найчастіше, чим більше витрачено

часу на розробку, тим більше грошей втратить ініціатор процесу розробки (замовник). У більшості випадків саме час є визначальним фактором при розробці програмного продукту.

Критерій «Економія енергії» вимірюється в процентах – результат застосування стратегії до конкретного програмного продукту, різниця між споживаної раніше і споживаної зараз енергією. Так як в залежності від оптимізацій швидкість виконання операцій програмного продукту буде змінюватися, вимірювати різницю енергоспоживання буде неможливо, має сенс в якості параметрів для порівняння використовувати площу фігури відсіченою відрізками і кривої енергоспоживання

$$S = \int_{t_1}^{t_2} f(t)dx,$$

де t_1 – початок вимірювання,

t_2 – кінець вимірювання.

Так як дана формулу зручно використовувати тільки тоді коли є рівняння залежності, то буде використаний чисельний метод розрахунку інтегралів методом трапецій:

$$S = \int_{t_1}^{t_2} f(t)dx = \frac{f(t_i)+f(t_{i+1})}{2}(t_{i+1} - t_i)$$

Результат впровадження тієї чи іншої стратегії тоді буде виглядати наступним чином:

$$D = (1 - \frac{S_c}{S_б}) * 100\%$$

де D – зміна енергоспоживання між тим, що було і тим, що стало в процентах,

$S_б$ – результат базового вимірювання,

S_c – результат вимірювання після оптимізації.

Обмеження «Глобальність змін» визначається кількістю компонентів системи (програмного продукту) зазнали зміни і тим самим характеризує зростання витрат на регресивний тестування (таблиця 3.1).

Зручність для користувача інтерфейсу та обмеження – класифікація за зручністю призначеного для користувача інтерфейсу описана в таблиці 3.2. Застосування деяких стратегій оптимізації програмного забезпечення по енергоспоживанню може спричинити за собою зміни в інтерфейсі і позначитися на його зручність.

Таблиця 3.1 – Види зміни по глобальності

Тип зміни	Опис
3 (Зміни в класі).	Зміни носять локальний характер і зачіпають один клас.
2 (Зміна в компоненті (групі класів)).	Зміни зачіпають групу класів, яка виконують одну глобальну роль.
1 (Зміни в усьому програмному продукті).	Зміні піддаються багато компонентів програмного продукту.

Таблиця 3.2 – Класифікація за зручністю призначеного для користувача інтерфейсу

Характеристика зручності	Опис
4 (Зручно)	Користувач задоволений, взаємодія з програмним продуктом не викликає складності.
3 (Злегка незручно)	Користувач може взаємодіяти з програмним продуктом, але це викликає легке роздратування.
2 (Незручно)	Користувач все ще може користуватися програмним продуктом, але це викликає сильне роздратування
1 (Дуже незручно)	Користувач не може користуватися програмним продуктом або може, але це пов’язане з критичним роздратуванням.

Існують також інші критерії, за якими можна оцінити доцільність впровадження тієї чи іншої стратегії оптимізації, але для наших цілей достатньо

об’єктивним буде визначення доцільність по вже перерахованим критеріям. Кожен з цих критеріїв має свій пріоритет як показано в таблиці 3.3.

Таблиця 3.3 – Пріоритети критеріїв для визначення доцільності впровадження оптимізації в роботу програмного забезпечення

Критерій	Пріоритет	Опис
Економія енергії	2	Має високе значення, так як є кінцевим результатом внесення оптимізації.
Трудовитрати	4	Мають найвищих пріоритетів, так як нам важливо, в першу чергу, закінчити програмний продукт раніше конкурентів або укластися в термін.
Глобальність змін	1	Від глобальності змін залежить час, які доведеться витратити на тестування.
Зручність для користувача інтерфейсу	3	Дуже важливий параметр, так як будь-який програмний продукт розробляється для кінцевого користувача.

Результатом застосування стратегії стане звіт, який міститиме об’єктивні чисельні показники, що відображають зміни за всіма визначальними критеріями. Але об’єктивний результат, по одній стратегії для одного програмного продукту, складно порівняти з об’єктивним результатом, по інший стратегії для іншого програмного продукту, в зв’язку з наявністю великої кількості критеріїв для порівняння (розмір програмного продукту, складність і якість коду та ін.).

3.2 Алгоритм обробки результатів

Які не вказані в визначають критерії в даному експерименті. Тому як рішення об’єктивний результат по конкретної стратегії буде переведений в стандартизований суб’єктивний результат, а кінцеві суб’єктивні результати

будуть порівняні між собою для визначення кращої стратегії для конкретного типу програмного продукту.

Вибірка типових додатків для формування стратегій щодо оптимізації ПЗ по енергоспоживанню. Для отримання досить точного результату при кінцевих витратах часу було прийнято рішення проводити експеримент на одному проекті:

Тестовий додаток `DDMailSender`**.** Для відправки картинок на пошту з підтриманням черзі відправки і збереженні її стану в базу даних, написане з використання модуля відправки DDSender.

У даному розділі були описані основні методи оптимізації по енергоспоживанню ПЗ, були розглянуті види акумуляторів використовуваних в мобільних пристроях. Так само в даному розділі вказано план практичної складової даного дослідження, який включає наступні етапи:

- написання тестового додатка середньої якості, без впровадження оптимізацій по енергоспоживанню;
- перевірка всіх викладених у розділі 1 методів оптимізації;
- складання звіту про результати та ефективності кожного з методів щодо застосування;
- аналіз отриманих результатів, побудова графіків, що відображають всі характеристики;
- складання стратегій і можливе написання програмних модулів (якщо рішення даної проблеми типово);
- складання кінцевого документа зі стратегії;
- кінцевий аналіз дослідження.

Як додаток для експерименту було використано тестове додаток DDMailSender, додаток для відправки графічних даних на сервер з використанням черги відправки і збереженням стану в базу даних.

Для оцінки ефективності перевірки поліпшень були вироблені критерії і обмеження для об’єктивної оцінки:

- економія енергії (критерій), має високе значення, так як є кінцевим результатом внесення оптимізації;
- трудовитрати (обмеження), мають найвищих пріоритетів, так як нам важливо, в першу чергу, закінчити програмний продукт раніше конкурентів або укластися в термін;
- глобальність змін (обмеження), від глобальності змін залежить час, які доведеться витратити на тестування;
- зручність для користувача інтерфейсу (обмеження), дуже важливий параметр, так як будь-який програмний продукт розробляється для кінцевого користувача.

3.3 Базові алгоритми роботи програмного продукту

Хоч програмний продукт є тестовим (написаний для цілей експерименту), але завдання, які він вирішує, зустрічаються в реальних проектах і є досить складними, і це робить його кандидатом для впровадження оптимізацій по енергоспоживанню.

DDMailSender призначений для підтримки черги і послідовної відправки даних (текстової інформації і графічних файлів) на електронну пошту. Так само є функція збереження в базу даних для можливості відновлення відправки якщо

додаток завершило роботи і функція зберігання історії вдалих і не вдалих відправок для повторного відправлення, уже відправлених даних.

До функціональності відправки в зв’язку з її можливим навантаженням, роботою з багатьох потоків і поновлювані представлені наступні вимоги:

- відсутність впливу на швидкодію UI інтерфейсу;
- підтримання історії відправок;
- оптимальна робота з ресурсами (для запобігання виходу з програми через брак оперативної пам’яті);
- возобновляемість при екстреному завершенні програми;
- робота в фоновому режимі (коли додаток згорнуто);
- підтримка послідовної і контрольованою відправки.

В рамках мобільних пристроїв оптимальне рішення даної задачі є досить складним у зв’язку з відносно невеликою потужністю пристрою і кількістю оперативної пам’яті. Нижче (рисунок 3.1) описаний результуючий алгоритм. Так само варто відзначити, що на кожному етапі зміни стану даних (почалася відправка, відправка завершена і т.ін.) Відбуватиметься повідомлення інших компонентів системи через центр нотифікацій.

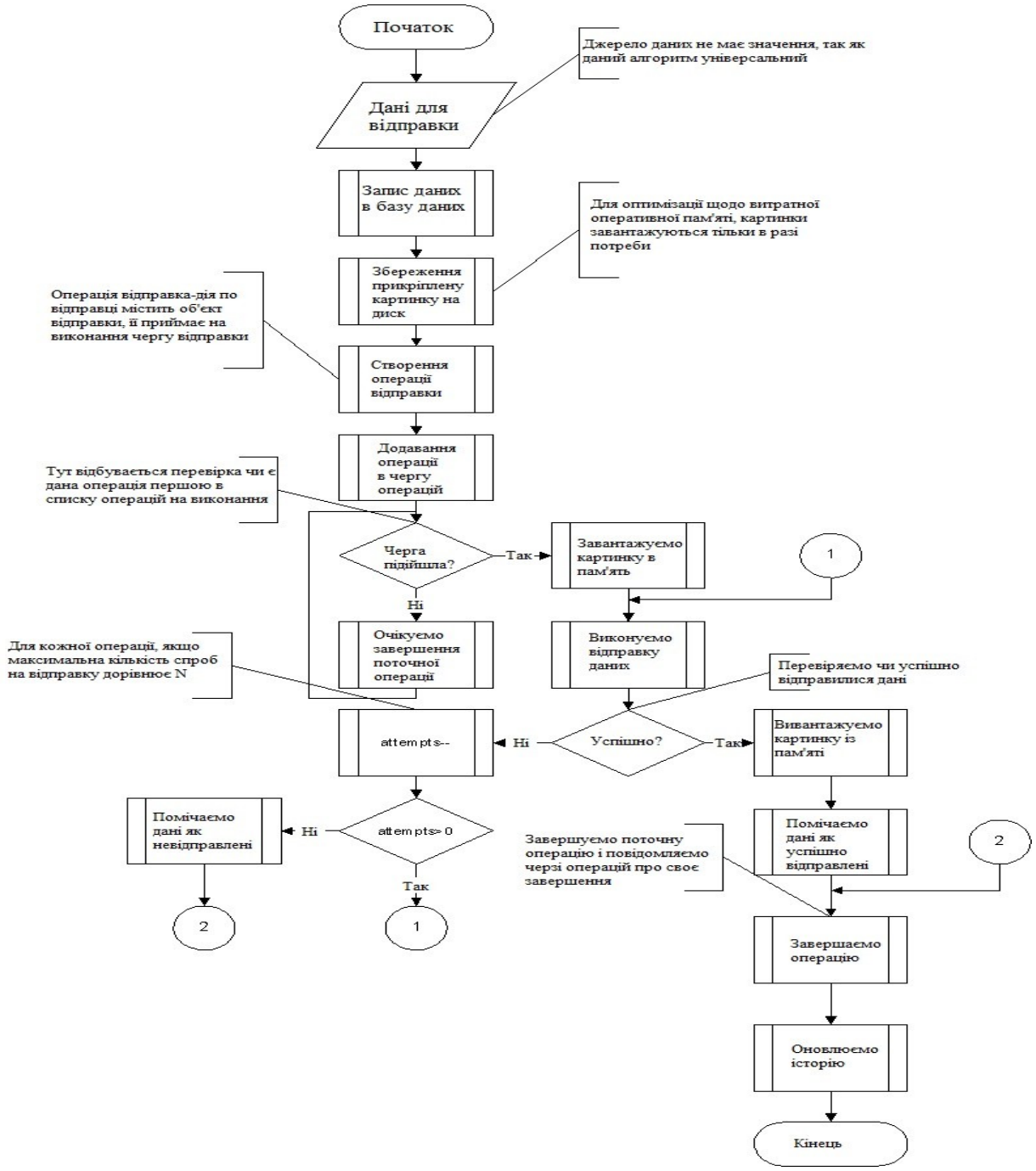

Рисунок 3.1 – Алгоритм додавання даних на відправку в чергу

У зв’язку з можливістю звернення до менеджера по роботі з базою даних з різних потоків, було прийнято рішення створити окрему чергу операцій для управління зверненнями до бази даних (як засіб по роботі з базою даних обраний фреймворк CoreData, основним недоліком, якого є специфічна робота в додатку). Загальний алгоритм роботи збереження змін в базу даних представлений в діаграмі (рисунок 3.2).

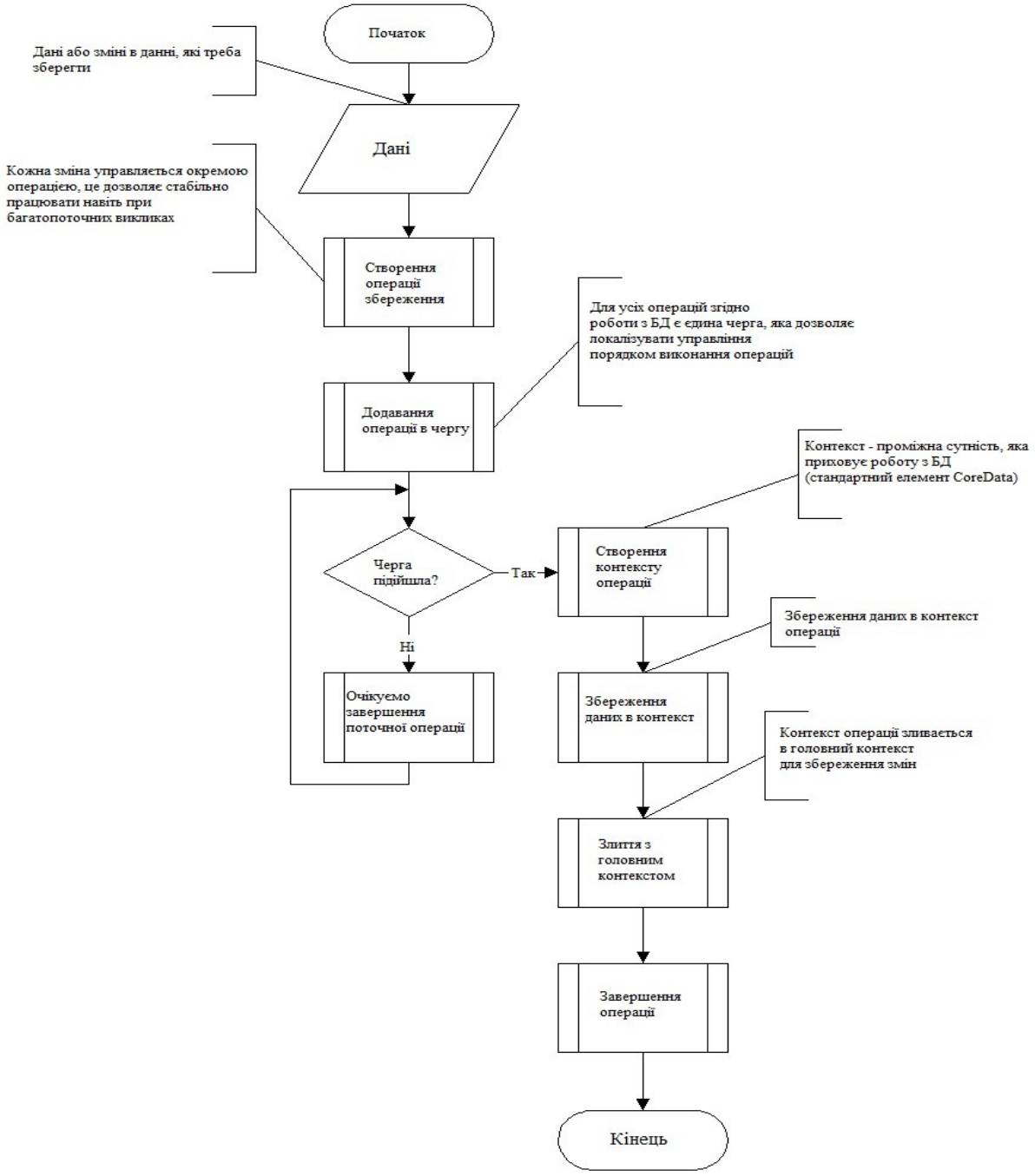

Рисунок 3.2 – Алгоритм збереження в базу даних

Для виключення заповнення всього вільного місця на диску, історія відправок має кінцевий розмір (N) і при кожному оновленні історії відбуватися перевірка кількості записів і видалення старих (рисунок 3.3).

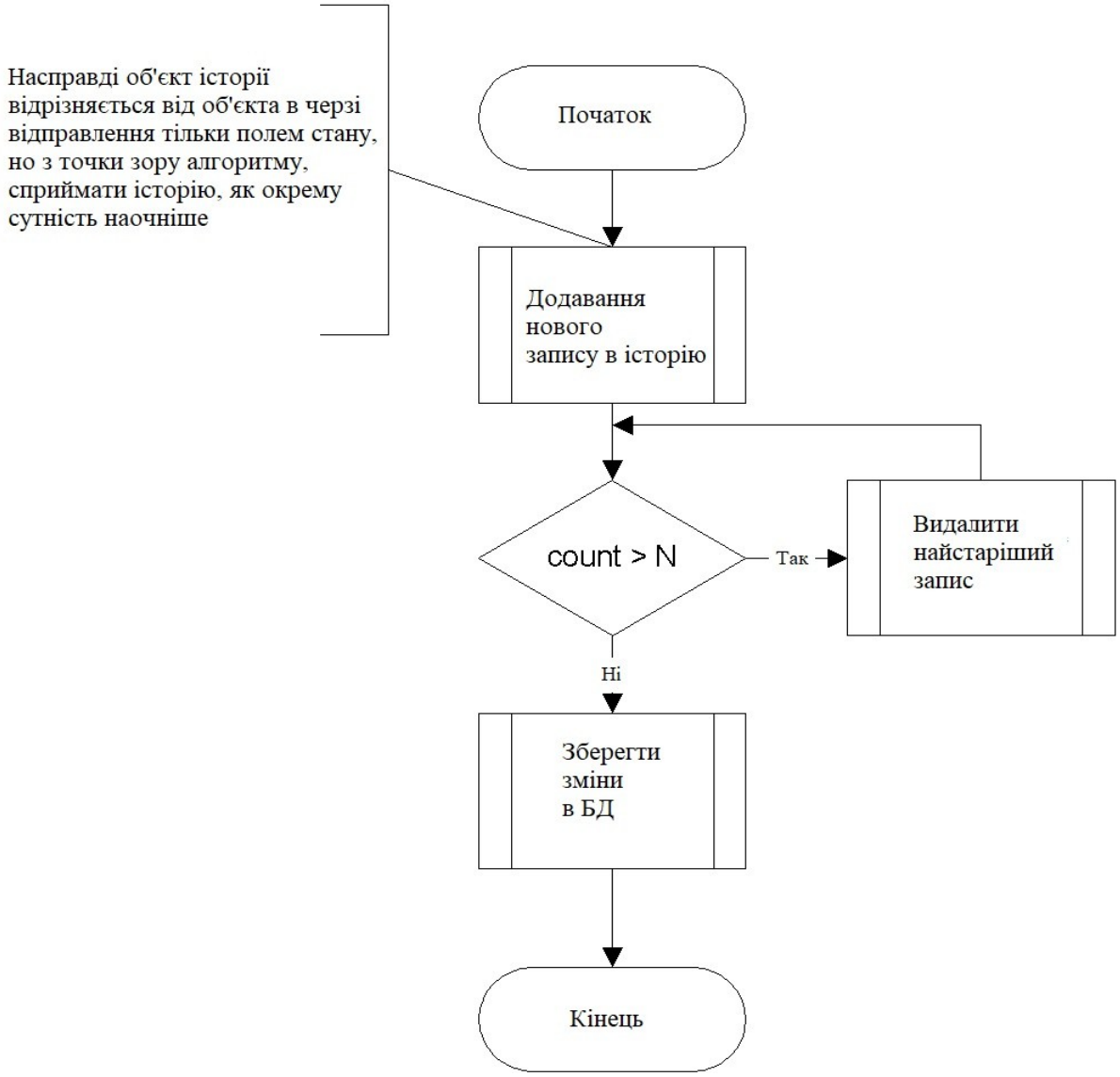

Рисунок 3.3 – Алгоритм поновлення історії відправок

3.4 Вибір і обґрунтування засобів розробки

В даний час розвиток програмних засобів здійснюється за рахунок автоматизації виконання таких стандартних операцій як: створення інтерфейсу, передача управління в залежності від стану системи , обробка виняткових ситуацій, створення моделі бази даних і запитів до неї і т.д. До основних характеристик сучасних засобів розробки програмного забезпечення відносять:

- використання CASE технологій;

- забезпечення доступу до бази даних;

- наявність візуальної технології розробки інтерфейсу;

- підтримка об’єктно-орієнтованого стилю програмування;

- наявність інструментальних засобів розробки додатків баз даних використовують реляційну модель даних;

- використання різних методів «візуалізації» як моделі даних, так і самих даних;

- надання коштів синхронізації і контролю версій складових частин проекту. Ці кошти використовуються при розробці програмного забезпечення групами програмістів;

- створення інсталяційних пакетів для поширення, розробленого програмного забезпечення.

При створенні програмного продукту основними критеріями вибору засобів розробки були:

– зручність використання;

– швидкість розробки додатків і роботи програми.

Як додаток до перерахованих засобів, можна вказати, що час розробки залежить від наданого інструментарію, наявності попереднього досвіду у розробників в використання відповідних програмних засобів і наявності, необхідних для їх ефективного використання апаратних засобів. Забезпечити мінімальний час розробки можна тільки при виконанні цих умов.

На даний момент особливою популярністю користуються такі мобільні операційні системи Android та iOS.

Для порівняння цих операційних систем скористаємося методом варіантних мереж. Оцінено по п’яти бальній шкалі наступні характеристики:

- продуктивність (4);

- зручність розробки (5);
- популярність (4);
- перспективність (3).

В круглих дужках вказані коефіцієнти вагомості, тобто важливості, кожної характеристики, визначені методом експертних оцінок.

Рішення поставленого завдання вибору програмного забезпечення методом варіантних мереж показано в таблиці 3.5.

Таблиця 3.5 – Вибір інструментального середовища методом варіантних мереж

Характеристика / Операційна система	1 (4)	2 (5)	3 (4)	4 (3)	Разом
Android	3	3	5	5	62
iOS	5	5	4	5	71

З результатів аналізу, проведеного методом варіантних мереж, слід, що кращою операційною системою для розробки є iOS.

Серед програмних засобів розробки, які мають характеристики необхідними для мобільного програмного продукту під iOS, доступними є наступні:

– Objective C.
– C ++.
– Ruby (rhomobile).
– JavaScript (багато кроссплатформенних фреймворків).

Для порівняння цих програмних продуктів використано методом варіантних мереж. Оцінимо за п’ятибальною шкалою наступні характеристики програмних продуктів:

- продуктивність (4);
- документованість (3);
- швидкість розробки програмного продукту (4);

- вимоги до апаратних ресурсів (3).

В круглих дужках вказані коефіцієнти вагомості, тобто важливості, кожної характеристики, визначені методом експертних оцінок.

Рішення поставленого завдання вибору програмного забезпечення методом варіантних мереж показано в таблиці 3.6.

Таблиця 3.6 – Вибір інструментального середовища методом варіантних мереж

Характеристика / Засіб розробки	1 (4)	2 (3)	3 (4)	4 (3)	Разом
Objective C	5	5	5	5	70
C ++	5	5	3	5	62
Ruby	2	3	3	3	38
JavaScript	3	3	4	4	49

З результатів аналізу, проведеного методом варіантних мереж, слід, що найкращим мовою розробки є Objective C.

4 ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Архітектура ПЗ, що розробляється

Додаток `DDMailSender` – призначений для відправки текстових даних і зображень на пошту, підтримує організацію черги вищезазначених даних, збереження свого стану в базу даних і роботу в фоновому режимі.

Нижче наведені діаграми класів, що описують основні взаємозв'язки в модулі, що розроблений. Дані діаграми описую основні клас модуля, в них не вказані інтерфейси, побічні класи та ін. (рисунок 4.1).

На даний момент вся робота з відправкою прихована за класом DDMailDispatcher, він дає дані внутрішньої черги повідомлень, відправляє на збереження дані в DDDatabaseManager, а також відправляє повідомлення про стані відправки в DDMailDispatcherNotificationManager, для їх подальшої розсилці по класах з необхідною інформацією.

В даному розділі описані всі класи DDMailSender і їх основні методи.

`DDClassDictionary` – клас словник, допоміжний клас для породжує фабрики `DDDataObjectCreator`.

Він зберігає в собі відповідності класів, де один клас є ключем, а інший значенням:

- `(Class) classForKey: (Class) aKey` – метод повертає клас відповідає класу, переданому як ключу;

- `(void) setClass: (Class) classObject forKey: (Class) aKey` – метод задає відповідність класів.

Клас `DDCoreDataOperation` – операція збереження, спадкоємець NSOperation доповнює його менеджментом допоміжного контексту CoreData:

`(id) initWithMainManagedObjectContext: (NSManagedObjectContext *) context withBlock: (void (^) (NSManagedObjectContext * context)) block`

метод конструктор, що приймає як параметр контекст операції збереження і блок (анонімна функція) який спрацьовує при виконанні цієї операції.

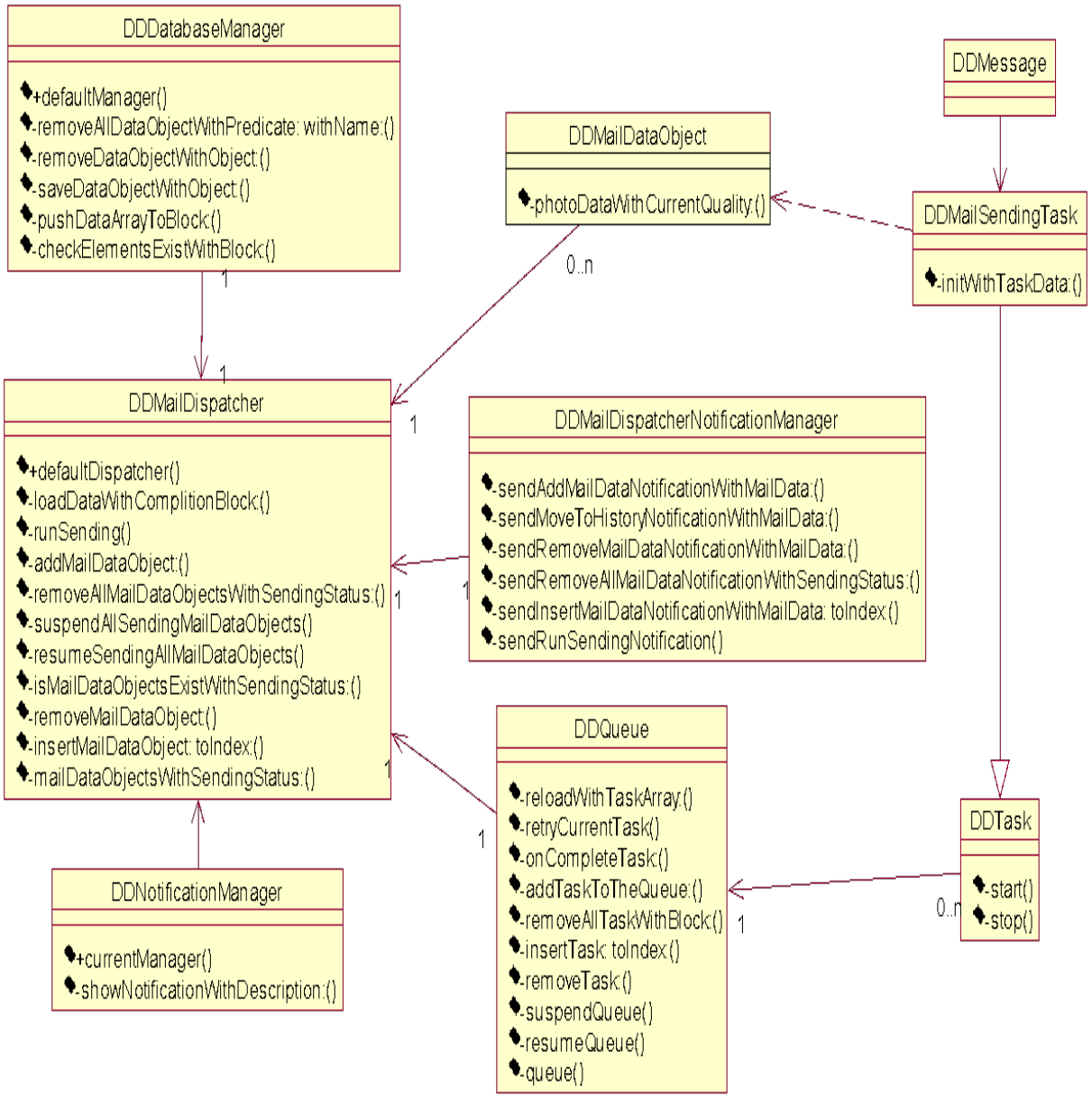

Рисунок 4.1 – Діаграма класів (DDMailDispatcher і базові компоненти відправки)

Головний клас менеджер `DDDatabaseManager` – одинак**,** для асинхронної роботи з базою даних.

- *`(DDDatabaseManager *) defaultMananger`* – метод, що дає доступ до статичного примірнику DDDatabaseManager.

- `(void) removeAllDataObjectWithPredicate: (NSPredicate *) predicate withName: (NSString *) name` – метод, що видаляє всі об'єкти і бази даних з ім'ям name і відповідні предикату.

- `(void) removeDataObjectWithObject: (DDDataObject *) dataObject` – метод, що видаляє об'єкт CoreData який відповідає модельному об'єкту dataObject.

- `(void) saveDataObjectWithObject: (DDDataObject *)dataObject` – метод, який шукає відповідний модельному дата об'єкту `CoreData` об'єкт і оновлює його з dataObject або створює порожній CoreData об'єкт і наповнює його з `dataObject`, а потім зберігає.

- `(void) pushDataArrayToBlock: (void (^) (NSArray * arrayOfData)) block withPredicate: (NSPredicate *) predicate withName: (NSString *) name` – асинхронний метод повертає в блок масив об'єктів відфільтрованих по предикату і з ім'ям моделі name.

- `(void) checkElementsaExistWithBlock: (void (^) (BOOL isExist)) block withPredicate: (NSPredicate *) predicate withName: (NSString *)name` – перевіряє на наявність елементів в базі даних відповідних предикату і з модельним ім'ям name.

`DDDataObjectCreator` – одинак, клас фабрика створює на базі CoreData об'єктів модельні об'єкти (DDDataObject).

`+ (DDDataObjectCreator *) sharedDataObjectCreator` – метод дає доступ до статичного примірнику класу DDDataObjectCreator.

`DDFileManager` – клас файловий менеджер для збереження, завантаження і видалення картинок з диска.

`+ (DDPhotoFileObject *) photoFileObjectWithSavingData: (NSData *) sendingData` – статична функція зберігає фотографію на диск і повертає `DDPhotoFileObject` з посиланням на неї.

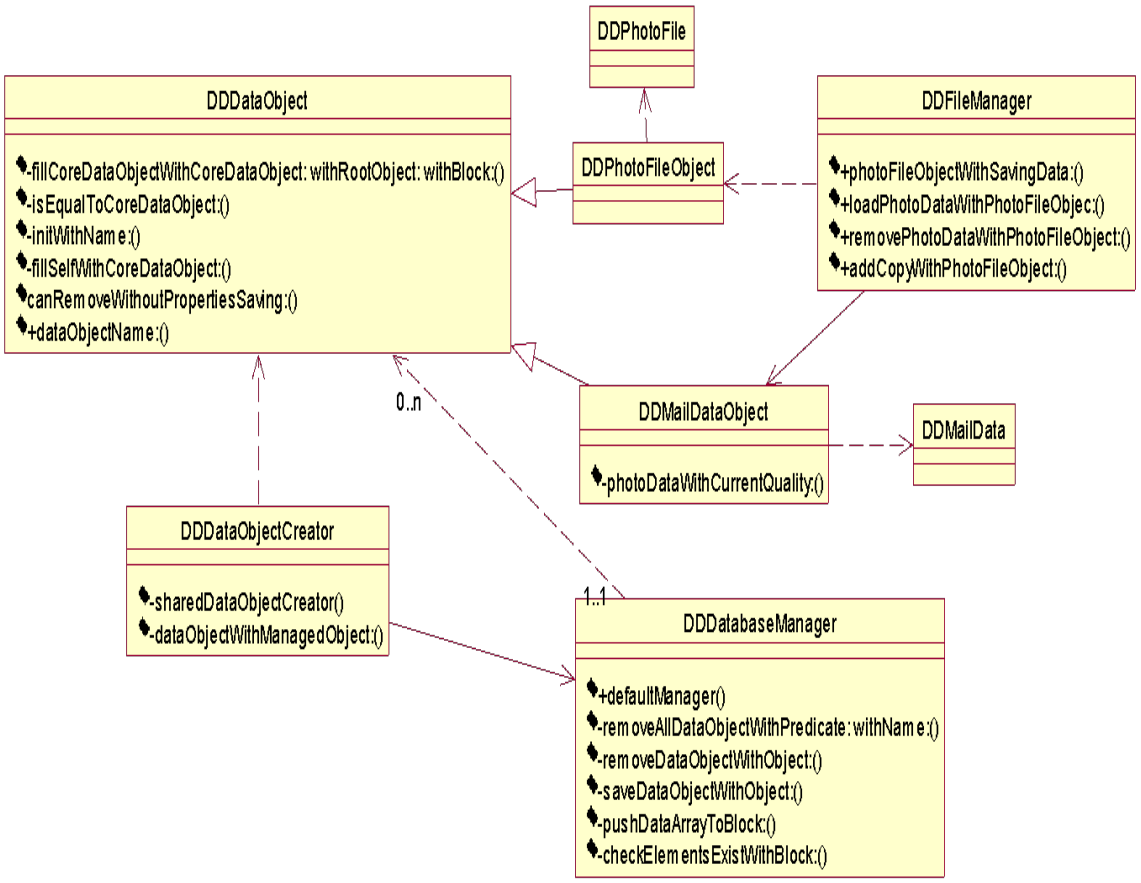

Рисунок 4.2 – Діаграма класів DDDatabaseManager і основні модельні дані

`+ (NSData *) loadPhotoDataWithPhotoFileObject: (DDPhotoFileObject *) photoFileObject` – метод, що завантажує в пам'ять фотографію і повертає її, як параметр приймає `DDPhotoFileObject` посилається на завантажену фотографію.

Метод метод, що видаляє DDPhotoFileObject з бази даних і відповідну фотографію з диска:

`+ (BOOL) removePhotoDataWithPhotoFileObject: (DDPhotoFileObject *) photoFileObject.`

`+ (Void) addCopyWithPhotoFileObject: (DDPhotoFileObject *) photoFileObject` – метод, який додає ще одне

посилання на фотографію яка вже збережена на диск, використовується якщо кілька моделей містять одну й ту ж саму фотографію.

`DDBackgroundTaskDispatcher` – базовий клас, інкапсулює в собі роботу з фоновими завданнями.

`- (void) stopBackgroundTask` – метод, який зупиняє фонову задачу.

`- (void) startBackgroundTask` – метод, що запускає фонову задачу.

`DDMailDispatcher` – одинак, головний клас компонента Sender, спадкоємець `DDBackgroundTaskDispatcher`, дозволяє додавати дані на відправку, а так само маніпулювати їх порядком, видаляти, зупиняти, поновлювати та _ор.

`+ (DDMailDispatcher *) defaultDispatcher` – метод, що дає доступ до статичного примірнику класу `DDMailDispatcher`.

- (void) loadDataWithComplitionBlock_void (^) ()) block – метод, який викликається один раз при старті програми, відновлює чергу з бази даних і готує її до роботи.

- (void) runSending – метод, що викликається після підготовки черзі до запуску, він запускає відсилання черзі.

- (void) addMailDataObject: (DDMailDataObject *) mailDataObject – метод, який додає дані в чергу на відправку.

- (void) removeAllMailDataObjectWithSendingStatus: (DDMailDataObjectSendingStatus) status – метод, що видаляє всі об’єкти з черги і бази даних з певним статусом.

- (void) suspendAllSendingMailDataObjects – метод, який призупиняє відправку даних.

- (void) resumeSendingAllMailDataObjects – метод, що відновлює відправку даних.

- (BOOL) isMailDataObjectsExistWithSendingStatus: (DDMailData ObjectSend ingStatus) status – метод, який перевіряє чи даних із зазначеним статусом в черзі.

- (void) removeMailDataObject: (DDMailDataObject *) mailDataObject – метод, що видаляє зазначені дані з черги.

- (void) insertMailDataObject: (DDMailDataObject *) mailDataObject toIndex: (int) index – метод, який додає дані в чергу за вказаною індексу.

- (NSArray *) mailDataObjectsWithSendingStatus:

(DDMailDataObjectSendingStatus) status – метод, який повертає масив даних з черги по зазначеному статусу.

DDMailDispatcherNotificationManager – клас що повідомляє додаток за допомогою нотифікацій про зміну стану черги відправки даних.

- (void) sendAddMailDataNotificationWith:

(DDMailDataObject *) mailDataObject – метод, що відсилає повідомлення про додавання даних в чергу.

- (void) sendMoveToHistoryNotificationWithMailData:

(DDMailDataObject *) mailDataObject – метод, що відсилає повідомлення про переміщення даних в історію.

- (void) sendRemoveMailDataNotificationWithMailData:

(DDMailDataObject *) mailDataObject – метод, що відсилає повідомлення про видалення даних з черги.

- (void) sendRemoveAllMailDataNotificationWithSendingStatus:

(DDMailDataObjectSendingStatus) status – метод, що відсилає повідомлення про видалення всіх даних з черги з певним статусом.

- (void) sendInsertMailDataNotificationWithMailData:

(DDMailDataObject *) mailDataObject toIndex: (int) index – метод, що відсилає повідомлення про додавання даних в чергу за індексом.

- (void) sendRunSendingNotification – метод, що відсилає повідомлення про початок відправки даних чергою.

DDMailSendingTask – клас операція відправки, спадкоємець DDTask.

- (id) initWithData: (id) taskData – конструктор, не започатковано операцію відправки з даними для відправки.

DDQueue – клас чергу, керує операціями класу DDTask. Дає можливість переміщати, видаляти, додавати та _op. Дії над операціями.

- (void) reloadWithTaskArray: (NSArray *) taskArray- метод, що відновлює чергу операцій з отриманого масиву

- (void) retryCurrentTask – метод, що запускає повторно поточну операцію.

- (void) onCompleteTask: (DDTask *) task – метод, завершальний зазначену операцію.

- (void) addTaskToTheQueue: (DDTask *) task – метод, який додає операцію в чергу.

- (void) removeAllTaskWithBlock: (void (^) (DDTask * task)) block – видалять всі операції, з черги попередньо викликаючи для кожної з них вказаний блок.

- (void) insertTask: (DDTask *) task toIndex: (int) index – метод, який додає операцію в чергу за індексом.

- (void) removeTask: (DDTask *) task – метод, що видаляє операцію з черги.

- (void) suspendQueue – метод, який призупиняє роботу черги.

- (void) resumeQueue – метод, що відновлює роботу черги.

- (NSArray *) queue – метод, який повертає всі операції в черзі.

DDTask – клас операція, базовий клас операція який приймає DDQueue.

- (void) start – метод запускає операцію на виконання.

- (void) stop – метод зупиняє виконання операції.

DDMessage – клас займається відправкою даних на пошту.

- (id) initWithMailData: (DDMailDataObject *) mailData – конструктор не започатковано клас даними на відправку.

- (void) sendMessage – метод, який розпочинає відправку даних на пошту.

- (void) stop – метод, що перериває відправку даних на пошту.

DDNetworkManager – одинак, клас для перевірки стану мережі.

- (NSUInteger) timeoutForCurrentConnectionType – метод, який повертає максимальний інтервал очікування відповіді для даного типу з’єднання.

- (void) checkConnectionWithDelegate: (id <DDNetworkManagerDelegate>) newDelegate – метод, що викликає метод делегата з результатом перевірки є з'єднання з інтернетом чи ні.

DDNotificationManager – одинак, клас керуючий показом повідомлень візуально схожих на повідомлення нотифікацій.

+ (DDNotificationManager *) currentManager – метод, що дає доступ до статичного примірнику класу.

- (void) showNotificationWithDescription: (DDNotificationManagerDescription *) messageDescription – метод, який показує повідомлення, дані якого заповнюються з messageDescription.

- (void) handleReceivedNotification: (UILocalNotification *) thisNotification – метод, що викликається при отриманні додатком локальної нотифікації.

DDPhotoFile – модельний клас фотографія CoreData, заповнюється з бази даних засобами CoreData.

DDMailData – модельний клас дані для відправки CoreData, заповнюється з бази даних засобами CoreData.

DDDataObject – абстрактний базовий клас моделі DDMailSender, його спадкоємці служать кінцевими моделями приховують роботу з CoreData. DDDataObject є базовим класом, з яким працює DDDatabaseManager, по суті, дата менеджер не знає про моделях, з якими працює.

- (void) fillCoreDataObjectWithCoreDataObject: `(NSManagedObject *) coreDataObject withRootObject: (NSManagedObject *) rootCoreDataObject withBlock: (void (^) (NSArray * arrayOfDataObjects, NSManagedObject * rootCoreDataObject)) block` – абстрактний метод, яким заповнюють об'єкт CoreData відповідно до свого стану і викликає блок з масивом подоб'єктов які треба заповнити.

- (BOOL) isEqualToCoreDataObject: (NSManagedObject *) coreDataObject- абстрактний метод, перевіряє відповідність даного об'єкта об'єкту CoreData.

- (id) initWithName: (NSString *) coreDataTypeName – абстрактний метод, инициализирует даний об’єкт модельним ім’ям CoreData моделі, з яких він буде згодом заповнюватися.

- (void) fillSelfWithCoreDataObject: (NSManagedObject *) coreDataObject – абстрактний метод, яким заповнюють поля об’єкта з відповідних полівcoreDataObject.

- (BOOL) canRemoveWithoutPropertiesSaving – абстрактний метод, перевіряє чи можна видалити об’єкт без пере збереження полів.

+ (NSString *) dataObjectName – повертає ім’я модельного класу CoreData з якого цей клас буде заповнюватися.

DDMailDataObject – клас спадкоємець DDDataObject, модель даних використовуваних в відправки і заповнюється з відповідного класу CoreData.

`DDPhotoFileObject` – клас спадкоємець DDDataObject, модель даних фотографія використовувана у відправці і заповнюється з відповідного класу CoreData.

`DDInfoView` – клас вікно (View) нотифікації.

DDTopViewInfo – клас одинак, керуючий поведінкою DDInfoView, що показує його і приховує.

+ (DDTopViewInfo *) defaultTopViewInfo – метод, повертає статичний екземпляр класу.

- (void) showMessageWithDescription:

(DDTopViewInfoDescription *) message – метод, який показує DDInfoView в верху екрана, заповненим з message.

(void) hide – метод, що приховує DDInfoView.

Для вимірювань енергоспоживання використовується Xcode instruments energy diagnostics (стандартна програма для дослідження енергоспоживання пристрою) [14].

Приклад роботи Xcode instruments energy diagnostics можна побачити на рисунку 4.3.

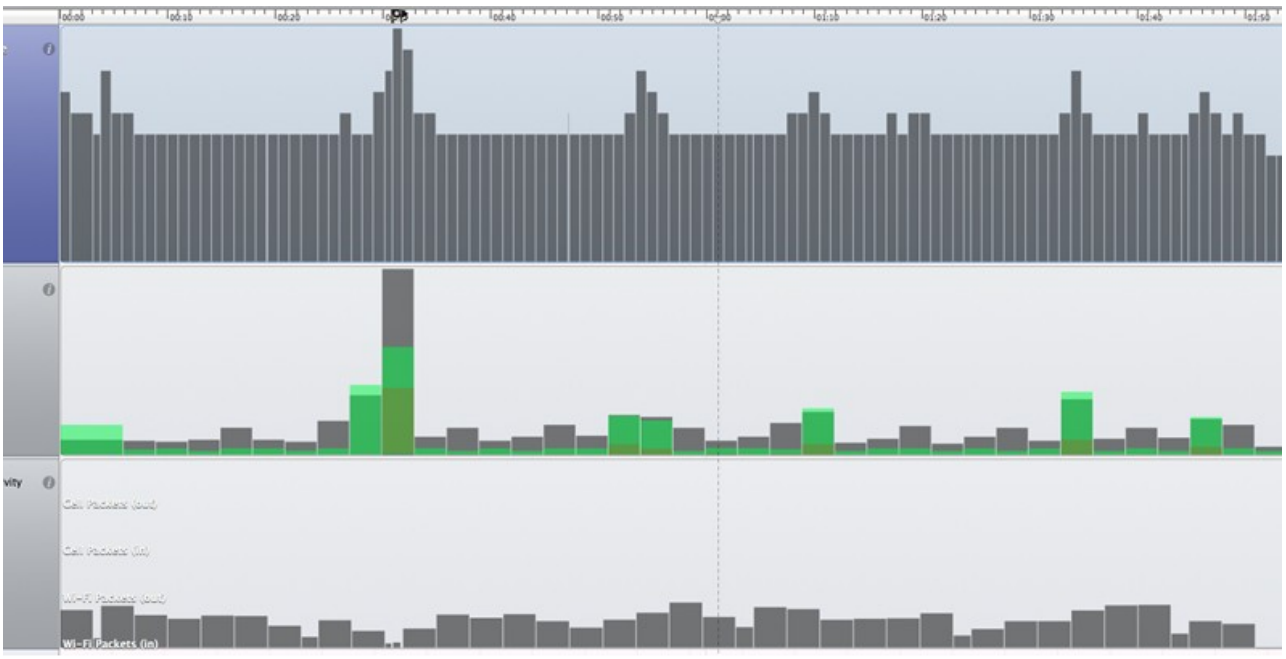

Рисунок 4.3 – Енергоспоживання, завантаження процесора і активність мережі пристрою без запущених додатків

В подальшому для кращої наочності будуть показуватися графіки, побудовані на основі даних одержуваних від інструментів (рисунок 4.4).

Рисунок 4.4 – Графік енергоспоживання пристрою без запущених додатків

Пристрій запитував поточного заряду батареї відбувається в середньому раз в секунду, значення, одержуване від пристрою, знаходиться в діапазоні від 0 до 20 і вимірюється в умовних одиницях EUL (EnergyUsage Level) [14]. На графіку «стрибкам» енергоспоживання відповідає згортання і розгортання меню налаштувань, а так же роботі Wi-Fi.

Як видно з вимірів середній показник витрати енергії 6.35 EUL, «стрибки» можна не враховувати, так як вони не впливають на загальну картину і статистично невідчутні.

4.2 Аналіз енергоспоживання пристрою

Що б оцінити ефективність внесених оптимізацій, були проведені заміри роботи програми DDMailSender без внесених оптимізацій. Експеримент складався з наступних кроків:

- запустити додаток;
- зупинити чергу відправки;
- додати 5 однакових картинок в чергу;
- запустити чергу і засікти час;
- почекати поки всі дані відправляться;
- засікти час закінчення відправки;
- закрити програму.

Для об’єктивності дослідження всі подальші вимірювання проводилися за цією ж схемою.

Як видно з екранній форми (рисунок 4.5) спочатку йде короткий пік, який відповідають запуску додатка, потім йде ще один, але більш тривалий, відповідний заповнення черги відправок і спад після нього відповідний закінченню запису на диск зображень.

Рисунок 4.5 – Графік енергоспоживання пристрою з активним застосуванням DDMailSender

4.3 Реалізація оптимізації для додатку DDMailSender

У зв'язку зі специфікою мобільної платформи і відштовхуючись від теоретичного матеріалу, обрано такі стратегії оптимізації:

- збільшення кількості потоків беруть участь у _ористувача інт.За рахунок збільшення завантаження процесора відправка виконається швидше, тим самим заощаджуючи енергію;

- мінімізація енергії – енергія економиться за рахунок мінімізації роботи з диском;
- потокове читання файлів з диска в процесі відправки даних. Енергія економиться за рахунок мінімізації використання оперативної пам’яті, картинка завантажується частинами в міру відправки.

Аналіз і реалізація оптимізації, що збільшує кількість потоків, які беруть участь у відправці. – дана оптимізація є найперспективнішою з погляду часу внесення змін і продуктивності, навіть за попередньою оцінкою. Вона полягає в збільшенні кількості потоків беруть участь у відправці, тим самим прискоривши процес відправки, але завантаживши процесор.

Результат впровадження даної оптимізації виглядає наступним чином (рисунок 4.4).

Таблиця 4.4 – Результат впровадження оптимізації збільшує кількість потоків беруть участь у відправці

	Значення
Трудовитрати	1 годину
Економія енергії щодо базового додатка	38%
Глобальність змін	1 клас
Зручність для користувача інтерфейсу	4 (Зручно)

Після внесення змін, відчутно прискорилася швидкість відправки даних, так само змінилася поведінка призначеного для користувача інтерфейсу, не завжди перший елемент в черзі встигає відправитися першим, але це ніяк не вплинуло на загальне враження від програми. Площа фігури відтятою відрізками часу (діапазон від 30 до 40 секунд) і кривої енергоспоживання дорівнює 138. Як і прогнозувалося, енергоспоживання зменшилася за рахунок збільшення кількості потоків.

Як показано на екранній формі (рисунок 4.6), поведінка енергоспоживання змінилося тільки в діапазоні відправки, а так все відповідає даним першого виміру без оптимізацій.

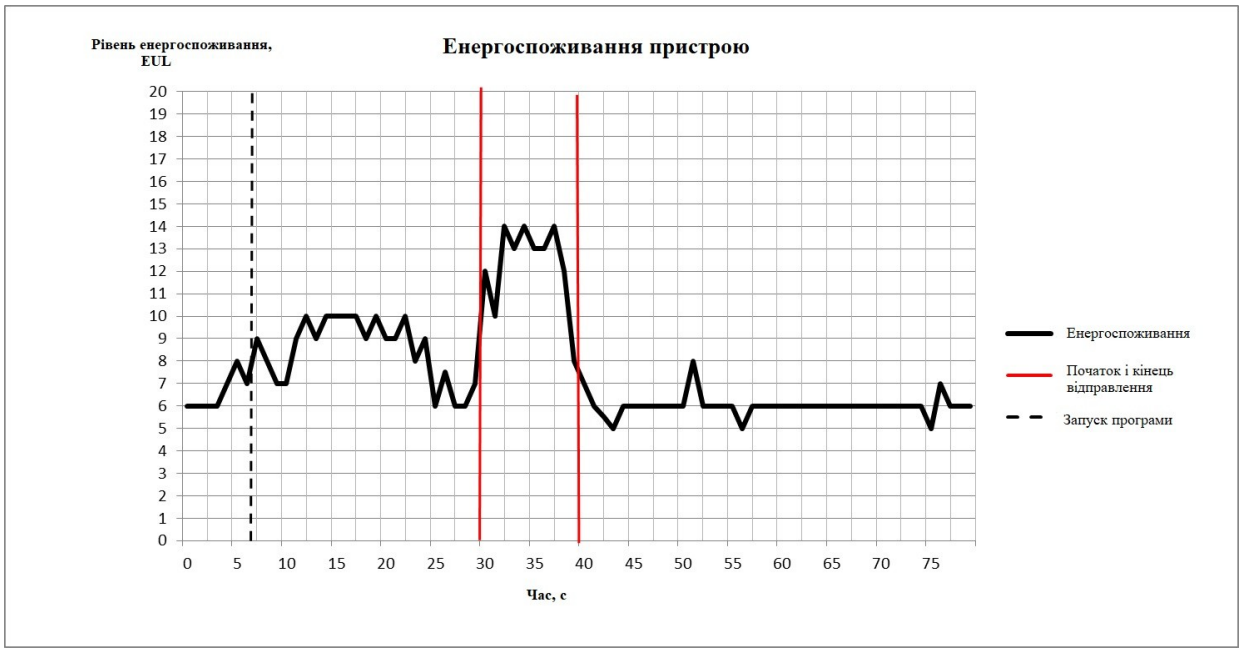

Рисунок 4.6 – Графік енергоспоживання пристрою з активним застосуванням DDMailSender (збільшено кількість потоків, що беруть участь у відправці)

Результат впровадження даної оптимізації виглядає наступним чином (таблиця 4.5).

Таблиця 4.5 – Результат впровадження оптимізації, які не вивантажуються графічні дані до їх остаточної відправки

Дія	**Значення**
Трудовитрати	12 годин
Економія енергії щодо базового додатка	9%
Глобальність змін	3 класу
Зручність для користувача інтерфейсу	1 (Дуже незручно)

Як видно з графіка (рисунок 4.7) при заповненні черзі енерговитрати зростають і не спадають, на відміну від минулих вимірів, так як графічні ресурси

не вивантажуються на диск, це впливає на зручність для користувача інтерфейсу (додаток може аварій про закритися).

Площа фігури відтятою відрізками часу (діапазон від 30 до 45 секунд) і кривої енергоспоживання дорівнює 202. Так само даний метод дає невеликий виграш в економії електроенергії, компенсуючи мінімізацію роботи з жорстким диском, великою витратою оперативної пам’яті.

Зміни торкнулися класи: DDPhotoFileObject, DDFileManager, DDMailSendingTask.

Рисунок 4.7 – Графік енергоспоживання пристрою з запущеним додатком DDMailSender (мінімізація дискових операцій)

Реалізація оптимізації, що реалізує потокову обробку даних є дуже перспективною, не тільки з точки зору оптимізації по енергоспоживанню, але і з точки зору роботи з великими файлами. Всі попередні оптимізації вирішували проблему роботи з великими файлами і могли аварійно завершитися, якщо графічний файл не зміг би помістится в оперативну пам’ять. Потокове читання дозволяє завантажувати файли в пам’ять шматками, тим самим мінімізуючи завантаження оперативної пам’яті, але підвищують навантаження на жорсткий диск.

Результат впровадження даної оптимізації виглядає наступним чином (таблиця 4.6).

Таблиця 4.6 – Результат впровадження оптимізації реалізує потокове читання і відправлення даних

	Значення
Трудовитрати	40 годин
Економія енергії щодо базового додатка	17%
Глобальність змін	4 класу
Зручність для користувача інтерфейсу	4 (Зручно)

Як видно з графіка (рисунок 4.8), заповнення черги йде, як і у випадку з _ористувача інтер додатком, але в процесі відправки графік енергоспоживання не піднімається вище 9 EUL. В даному випадку немає виграшу в швидкості відправки як з першої оптимізацією, але потоковая завантаження дозволяє зробити додаток більш універсальним і _ористувача ін.

Рисунок 4.8 – Графік енергоспоживання пристрою з запущеним додатком DDMailSender (потокове читання і відправка даних)

Зміни торкнулися класів: DDPhotoFileObject, DDFileManager, DDMailSendingTask, DDMessage.

5 ОПИС МОЖЛИВОСТІ ВИКОРИСТАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Для оцінки доцільності впровадження алгоритмів оптимізації, отриманий результат вимірювань треба порівняти, використовуючи критерії, описані в роботі.

Нижче наведена таблиця 5.1, в яку винесені всі результати вимірювань.

Таблиця 5.1 – Результати впровадження оптимізацій в програмний продукт

Номер оптимізації	Трудовитрати (_ори. Ч)	Економія енергії (%)	Глобальність змін (кількість класів)	Зручність для користувача інтерфейсу
+ 1	1	38	1	4 (Зручно)
2	12	9	3	1 (Дуже незручно)
3	40	17	4	4 (Зручно)

Як видно з таблиці найкращий результат показала перша оптимізація (оптимізація збільшує кількість потоків беруть участь у відправці), але в виходячи з табличних даних складно визначити яка оптимізація краще: друга або третя. Для порівняння цих оптимізацій скористаємося методом варіативних мереж, пріоритети для критеріїв, оцінювати потрібно за п’яти-бальною системою (_орис. 5.2).

За результатами вимірювань, найбільшою доцільністю впровадження володіє перша оптимізація, але з точки зору універсальності і стабільності роботи уваги заслуговує третя оптимізація, так як вона позитивно позначається на завантаженні оперативної пам’яті і дозволяє працювати з великими даними не заваетажуючи їх повністю в пам’ять. Друга оптимізація є не доцільною, так як погіршує загальну роботу програми.

Таблиця 5.2 – Вибір оптимізації методом варіантних мереж

	Трудовитрати (4)	Економія енергії (2)	Глобальність змін (1)	Зручність для користувача інтерфейсу (3)	Разом
1	5	5	5	5	50
2	4	3	4	1	29
3	3	4	3	5	38

Таким чином, були розглянуті і описані основні алгоритми функціонування прототипу додатку, в які будуть впроваджені оптимізації по мінімуму енергоспоживання.

Алгоритм додавання даних на відправку в чергу. В рамках додавання даних в чергу для відправки проводиться збереження в базу даних для можливості відновлення стану відправки при виході з; підготовка графічних даних до відправки, вивантаження і завантаження в пам’ять в залежності від стану операції відправки. Даний алгоритм є ключовим в роботі додатка і до нього пред’являються підвищені вимоги, так як нестабільна робота алгоритму додавання даних в чергу відправки може дестабілізувати роботу всієї системи.

Алгоритм збереження даних в базу даних. Даний алгоритм задіюється в алгоритмі додаванні даних в чергу на відправку, так як додаток працює з великою кількістю потоків, робота з базою даних організована за принципом черги операцій, це дозволяє управляти пріоритетом кожного зміни в базу даних і дає можливість стабільно працювати при великій кількості звернень з різних потоків.

Алгоритм поновлення історії відправок. Для надання можливості надсилання або перегляду вже відправлених даних в додатку підтримується історія відправок. Щоб обмежити розмір історії при додаванні нових записів відбувається фільтрація і видалення найстаріших записів, якщо кількість записів в історії перевищує ліміт.

Всі перераховані вище алгоритми є перспективними для впровадження оптимізації за критерієм мінімізації енергоспоживання.

Була описана архітектура для безпосереднього створення програмного забезпечення**.** Було виконано впровадження оптимізацій в алгоритми роботи програмного забезпечення та оцінено їх ефективність. За результатами експерименту побудовані графіки енергоспоживання і таблиці ефективності впроваджуваних оптимізацій.

ВИСНОВКИ

В епоху, коли швидко розвиваються інформаційні технології, постійно зростаючої кількості мобільних пристроїв і кількості ПЗ для них, особливо гостро виникає проблема швидкого розряду акумуляторів. Дана проблема з’явилася не тільки за межею технологій створення накопичувальних елементів акумуляторів, а й через неправильного підходу до створення ПЗ під мобільні платформи. У даній атестаційній роботі розглянуто способи мінімізації енергоспоживання шляхом оптимізації алгоритмів роботи ПЗ.

Розглянута проблематика сучасної розробки програмного продукту для мобільних пристроїв і вже існуючі стратегії оптимізації ПЗ. Так само були виділені ключові аспекти, на які треба звертати увагу при розробці екологічного ПЗ.

Були виділені критерії, за якими буде відбуватися оцінка внесених оптимізацій в програмного продукту:

- трудовитрати, час, що витрачається на впровадження даної функціональності в людино-годинах;
- різниця між енергією, що витрачається до оптимізації і після у відсотках;
- кількість компонентів, які були змінені в процесі оптимізації;
- зручність інтерфейсу користувача, враження від роботи з додатком, його зручність.

Розроблено і описано основні алгоритми, які використовувалися при розробці прототипу додатка для перевірки стратегій оптимізації програмного продукту.

Результати мають такий вигляд:

- збільшення кількості потоків беруть участь у відправці – 38% економії енергії при мінімальних витратах на впровадження;
- мінімізація дискових операцій – 9% економії енергії, погіршення роботи програми та зниження його стабільності;
- потокове читання файлів з диска в процесі відправки – 17% економії енергії, підвищення стабільності додатки.

Підбито підсумки проведення експерименту, в результаті яких була обрана найрентабельніша оптимізація зменшивши споживає на 38%.

Розроблені алгоритми оптимізації ПЗ дозволяють скоротити енергоспоживання мобільного пристрою.

Виходячи з результатів можна дати наступну рекомендацію – використовувати максимальну кількість потоків при великовагових операція для зниження енергоспоживання додатки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. 3-е изд. – К.: Наукова думка, 1989. – 376 с.

2. Горин А.К. Автоматизация проектирования структур данных АИС // Кибернетика. – 1989. – №6. – С. 114–117.

3. Дал У., Дейкстра Э., Хоар К. Структурное программирование. – М.: Мир, 1975. – 248 с.

4. Соболев В.Е., Терзян Т.К., Цейтлин Г.Е., Ющенко Е.Л. Р-синтезатор – инструментарий интегрированной технологии производства программ // УСиМ. – 2006. – №1. – С. 35–42.

5. Как программировать на XML / Х.М. Дейтел, П.Дж. Дейтел, Т.Р. Нието и др. – М.: Бином, 2011. – 944 с.

6. Яценко Е.А. Конструирование параллельных объектно-ориентированных программ // Проблемы программирования. Спец. выпуск по материалам 3-й Международной научно-практической конференции по программированию УкрПРОГ’2002. – К.: ИПС НАН Украины, 2016. – №1–2. – С. 188–197.

7. Дейкстра Э. Дисциплина программирования. – М.: Мир, 1978. – 274 с.

8. Цейтлин Г.Е., Яценко Е.А. Элементы алгебраической алгоритмики и объектно-ориентированный синтез параллельных программ // Математические машины и системы. – 2013. – №2. – С. 64–76.

9. Мальцев А.И. Итеративные алгебры и многообразия Поста // Избр. тр. А.И. Мальцева. – Т. 2. – 1976. – С. 316–330.

10. Жоголев Е.А. Гиперпрограммирование и базы прикладных программ // Программирование. – 2002. – №6. – С. 24–31.

11. Jackson M.A. Principles of program design. – London: Academic Press., 2007. – 299 p.

12. Jacobson I., Griss M., Johnson P. Software Reuse. – N.-Y.: Addison-Wesley, 2017. – 497 p.

13. Яценко Е.А. Алгебры гиперсхем и интегрированный инструментарий синтеза программ в современных объектно-ориентированных средах // Кибернетика и системный анализ. – 2014. – №1. – С. 47–52.

14. Фаулер М., Скотт К. UML. Основы: Пер. с англ. – СПб.: Символ-Плюс, 2012. – 192 с.

15. Lea D. Concurrent Programming in Java. – Reading: Addison-Wesley, 2017. – 339 p.

16. Letichevsky A. A., Gilbert D.R. A general theory of action languages // Cybernetics and Systems Analysis. – 2017. – №1 (36). – P. 16–36.

17. Letichevsky A.A., Gilbert D.R. Agents and environments // Proc. International scientific and practical conference on programming. – Kiev: Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine, 1998. – P. 32–44.

18. Холл М., Браун Л. Программирование для Web. Библиотека профессионала: Пер. с англ. – М.: Вильямс, 2012. – 1264 с.

19. Перевозчикова О.Л., Ющенко Е.Л. Диалоговые системы. – К.: Наукова думка, 2004. – 183 с.

20. IEEE Std. 610.12:1990. IEEE Standard Glossary of Software Engineering Terminology.

21. Основы инженерии качества программных систем / Ф.И.Андон, Г.И.Коваль, Т.М. Коротун, В.Ю. Суслов / Под ред. И.В. Сергиенко. – К.: Академпериодика. – 2012. – 504 с.

22. Парадигма качества программного обеспечения / Андон Ф.И, Суслов В.Ю., Коротун Т.М., Коваль Г.И. //Проблемы программирования. –2009. – №2. – С. 51-62.

23. Липаев В.В. Обеспечение качества программных средств. – М: СИНЕРГ. – 2011. – 380 с.

24. ДСТУ 3918-99 Інформаційні технології. Процеси життєвого циклу програмного забезпечення. – Київ. – Держстандарт України. – 2000. – 49 с.

25. Коваль Г.И., Коротун Т.М., Остапенко А.П. Превентивное тестирование и оценка надежности программного обеспечения как форма

управления риском проекта. //Сб. Программная инженерия. – Киев. – 1993. – С. 19-26.

26. Software Engineering Body of Knowledge (SWEBOK). // ISO/IEC JTC1/SC7 N2517. Software & System Engineering Secretariat, Canada, 2011. – 220 p.

27. Shostak I., Matyushenko I., Romanenkov Yu., Danova M., Kuznetsova Yu. Computer Support for Decision-Making on Defining the Strategy of Green IT Development at the State Level. In book: Green-IT Engineering: Social, Business and Industrial Applications, V. Kharchenko, Y. Kondratenko, J. Kacprzyk (Eds.), Vol. 171. Berlin, Heidelberg: Springer International Publishing, 533–559 (2018),