МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

"Високоефективне крос-платформне програмне забезпечення системи моніторингу стану акумулятору портативного пристрою з використанням фреймворку QT"

Завідувач випускаючої кафедри **Довбиш А.С.**

Студент групи ІНз-71с **Ахтирцев О.В.**

Керівник роботи **Прилепа Д.В.**

Суми, 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую _____
Зав. кафедри Довбиш А.С.
"_____" _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи __ спеціальності "Комп'ютерні науки" заочної форми навчання _____.

Тема: «Високоефективне крос-платформне програмне забезпечення системи моніторингу стану акумулятору портативного пристрою з використанням фреймворку QT»

Затверджена наказом по СумДУ
№_____ від _____2021 р.

Зміст пояснювальної записки: 1) огляд існуючих засобів 2) постановка задачі 3) розробка програмного забезпечення 4) короткий опис програмної реалізації 5) аналіз отриманих результатів

Дата видачі завдання "_____"_____2021 р.
Керівник випускної роботи _____ Прилепа Д.В.
Завдання прийняв до виконання _____

РЕФЕРАТ

Записка: 41 с., 5 рис., 11 літературних джерел, 2 додатки.

Об'єкт дослідження — кросплатформне програмне забезпечення, системне програмування.

Мета роботи — розробка мульти-платформного додатку, дослідження роботи програми на різних операційних системах.

Результати — розроблено кросплатформний додаток, що працює на операційних системах Windows та GNU/Linux.

КРОСПЛАТФОРМНИЙ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ОПЕРАЦІЙНА СИСТЕМА, ФРЕЙМВОРК, C++, C++11 QT, СИСТЕМНЕ ПРОГРАМУВАННЯ, WINDOWS, LINUX, ПОРТАТИВНИЙ ПРИСТРІЙ

ЗМІСТ

ВСТУП

Робота присвячена темі початкової розробки програмного забезпечення, що буде працювати не тільки на конкретній платформі, а на декількох одночасно, без суттєвої зміни самого програмного коду продукту.

Кроссплатформеність забезпечується за допомогою використання високорівневих мов програмування або середовищ виконання програм. Типовим прикладом кросплатформенного програмного забезпечення є додатки, які можуть однаково працювати як на операційній системі Windows, так і GNU/Linux.

Мета даної роботи – це не тільки досягти кросплатформеності, а й якомога більшої продуктивності.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

Сьогодні практично неможливо уявити собі додаток, що не володіє інтерфейсом користувача. Поняття *Software* (програмний продукт), *Apps* (Додатки) і *GUI* / (Graphical User Interface, графічний інтерфейс користувача) нерозривно пов'язані один з одним.

Хоча кожна з операційних систем володіє всім необхідним для створення графічного інтерфейсу користувача, використання цих доступних «інструментів» вимагає великих витрат часу і практичного досвіду. Навіть ті бібліотеки, створені для того, щоб полегшити процес написання програм, не дають процесу створення програм і додатків тієї про простоти і легкості, якої хотілося б розробникам.

Тому і сьогодні розробники як і раніше витрачають масу часу на реалізацію інтерфейсу користувача, але вже набагато ефективніше і з меншими зусиллями. Але найбільший недолік, пов'язаний із застосуванням таких бібліотек, - це *платформозалежність*.

Платформонезалежна реалізація додатків – це майбутнє програмної індустрії. З кожним днем з розвитком технологій вона набуває все більш наростаючого значення.

Саме такий підхід не дає залишити без уваги користувачів Mac OS X або мобільних пристроїв, що працюють на операційній системі Android, тільки тому що ви є розробником додатків тільки під конкретну операційну систему, Windows, Linux, тощо.

Дозволивши своєму додатку працювати на різних системах, можна спостерігати помітний приріст користувачів (клієнтів). Виграш від реалізації платформо незалежних додатків помітний одразу – значно скорочується час на розробку, так як Вам не потрібно адаптувати код багатократно під різноманітні ОС, під кожну платформу, і що не менш важливо, зникає необхідність знати нюанси кожної з платформ, для якої розробляється програмний продукт. Також немає

необхідності формувати спеціальні команди розробників для кожної платформи – це все може скоротити не тільки час на розробку, але і собівартість програмного продукту.

Разом з цим і помітно виросте якість програмного продукту, так як він буде тестуватися на декількох платформах одночасно, а помилки будуть виправлятись в одному і тому ж вихідному коді програми.

1.1 ОГЛЯД ІСНУЮЧИХ ЗАСОБІВ ДЛЯ СТВОРЕННЯ КРОС-ПЛАТФОРМНИХ ДОДАТКІВ

В якості платформонезалежних середовищ розглядалися й інші засоби розробки, проектування, і підтримки крос-платформних додатків, як на мові C++ [1], так і на інших мовах програмування, у тому числі інтерпретованих, наприклад JavaScript, Python. Але мови такого типу не гарантують нам високу ефективність роботи додатку, а також стабільність при виконанні програми. Для додаткових уникнення помилок при розробці, а також підтримці програмного продукту у подальшому, доцільніше було б використовувати мову з статичним типізуванням.

Отже, були розглянуті наступні засоби розробки, які використовують мову програмування C++:

1. бібліотека wxWidgets

Бібліотека wxWidgets[2] використовую рідні (native) елементи управління (controls) операційної системи. Також була портована на мову Python, Perl та Ruby. Але її недоліки полягають у тому, що дану бібліотеку не підтримують у новітньому стилі інтерфейсу (починаючи з 2010 року приблизно). Її стиль більше схожий на стиль застарілих операційних систем, наприклад, таких як Windows Vista, OS X версії 10. Також бібліотека wxWidgets[2] потребує більш глибокого розуміння, як працює цільова операційна система.

2. бібліотека GTK-3

Бібліотека має дуже простий API (програмний інтерфейс), схожий на Qt, також являється найпопулярнішим засобом для програмування під X Window System (ОС GNU/Linux). Додатки, які були розроблені з допомогою даної бібліотеки, також працюють на операційних системах Windows та Mac OS X. Але, як було вище сказано, дана бібліотека більш підходить для розробки під графічне середовище

GNOME операційної системи Linux, а для інших ОС, розробнику доведеться прикласти достатньо зусиль. [3]

3. CEF, Chromium Embedded Framework [4]

Даний фреймворк був розроблений компанією Google для операційних систем Windows, Linux та OS X. Із переваг даного засобу можна виділити високий контроль, зручність, якщо потрібно додати інтерфейс Web-браузера у додаток, на нього покладається маса проектів та/або інструментів, що дану платформу довговічною, підтримуваною. Але присутні недоліки, що змусять вас розглядати інші варіанти, тому що даний фреймворк є набагато складнішим за вище приведені засоби, а також недостатню початков базу для розробника, тому останнім доводиться підключати інші допоміжні бібліотеки.

1.2 ПОСТАНОВКА ЗАДАЧІ

Метою даної роботи являється розробка інформаційного та програмного забезпечення системи відображення поточного стану заряду акумулятору, що дає можливість визначити приблизний час до повної зарядки та/або розрядки акумулятору.

Для успішного виконання роботи потрібно виконати наступні задачі:

1. Сформувати концептуальну модель інформаційної системи.

2. Розробити алгоритми функціонування системи в режимах накопичення та візуалізації даних моніторингу.

3. Програмно реалізувати розроблені алгоритми.

4. Провести перевірку працездатності розробленого програмного забезпечення, а саме:

   4.1. можливість роботи на різних операційних системах;

   4.2. можливість конфігурування параметрів прозорості віджету, та закріплення над всіма вікнами;

   4.3. при наведенні курсору на віджет повинна бути можливість відображення інформації про стан акумулятору, приблизний час до повного заряджання/розряджання в залежності від того, чи підключена машина до електроспоживання;

   4.4. можливість змінювати розмір під час скролінгу.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

Для вирішення даної задачі, а саме: реалізації високоефективного прикладного програмного забезпечення, яке підходить для будь-якої платформи (або операційної системи) було прийнято рішення використати наступні технології:

- Мова програмування C++ (C++11);
- Крос-платформний фреймворк Qt версії 5.12 для реалізації програмного інтерфейсу, інтерфейсу користувача (UI).

Так як мова C++ є об'єктно-орієнтованою мовою, то краще було б слідувати основним "еталонам об'єктно-орієнтованого програмування", використовувати SOLID-принципи [5], принцип DRY (don't repeat yourself, - не повторюйся) а також патерни проектування.

SOLID - це акронім, літери яких розпочинають назви п'яти принципів розробки проектування в об'єктно-орієнтованому програмуванні - принципи єдиної відповідальності (Single responsibility), відкритості / закритості (Open-closed principle), підстановки Барбари Лісков (Liskov substitution), поділу інтерфейсу (Interface segregation) та інверсії залежностей (Dependency inversion).

Даний акронім був запропонований Робертом Мартіном (Robert Martin), автором кількох книг, широко відомих серед розробників. Принципи SOLID дозволяють будувати масштабовані програмні продукти зі зрозумілою бізнес-логікою, які легко підтримувати у майбутньому.

Single responsibility - принцип єдиної відповідальності.

Суть даного принципу полягає у тому, що кожен клас повинен мати один обов'язок, і цей обов'язок повинен бути інкапсульований у єдиний клас, а всі його сервіси та методи повинні забезпечувати виконання цього обов'язку.

Дотримання даного принципу полягає в основному у декомпозиції складних класів, що беруть на себе велику частину функціональності, на прості, відповідальність яких дуже спеціалізована.

Open-closed principle - принцип відкритості / закритості.

Принцип відкритості / закритості полягає в тому, що програмні сутності (класи, інтерфейси, модулі, функції і т. д.) повинні бути відкриті для розширення, але закриті для модифікації. Це означає, що ці сутності можуть міняти свою поведінку без зміни їх вихідного коду.

У цьому контексті мається на увазі, що відкритість для розширення сутності - це можливість додати для класу, модуля або функції нову поведінку, якщо необхідність в цьому виникне, а закритість для змін - це заборона на зміну вихідного коду програмних модулей. На перший погляд, це звучить доволі складно і суперечливо, але якщо розібратися, то принцип цілком логічний.

Liskov substitution principle - принцип підстановки Барбари Лісков.

Цей принцип у формулюванні Роберта Мартіна полягає в тому, що функції, які використовують базовий тип даних, повинні мати можливість використовувати підтипи базового типу, але не знаючи про це. Дотримання принципу полягає в тому, що при побудові ієрархій успадкування створювані класи-нащадки повинні коректно реалізовувати поведінку супер типу. Тобто якщо базовий тип реалізує конкретну поведінку, то вона повинна бути коректно реалізовано і для всіх його нащадків.

Interface segregation principle - принцип поділу інтерфейсів.

Суть принципу поділу інтерфейсів у тому, що класи-нащадки не повинні бути залежними від методів, які вони не використовують». Занадто “массивні” інтерфейси необхідно декомпозувати на більш дрібні і більш спеціалізовани, щоб класи, які розширюють малі інтерфейси, знали тільки про ті методи, які необхідні їм у роботі. Тобто зміна методу якогось інтерфейсу не повинна впливати на клієнтів, які цей метод не використовують.

Dependency inversion - принцип інверсії залежностей.

Модулі верхніх рівнів повинні бути незалежними від модулів нижніх рівнів, а обидва типи модулів повинні залежати від абстракцій; абстракції не повинні залежати від деталей, але ці деталі повинні залежати від абстракції.

Дотримання принципу *DRY* (don't repeat yourself / не повторюй себе) дозволяє досягти високої ефективності при супроводі програмного продукту: внесення змін і тестування якості продукту (quality assurance) стають значно простішими. Якщо код не дублюється, то для зміни логіки програми досить внесення виправлень всього в одному місці у вихідному коді. Також розробникам значно простіше тестувати одну, нехай і більш складну, функцію, а не набір з десятків з однаковим кодом. При проходженні DRY спрощується і повторне використання коду, винесеного зі складних алгоритмів, що дозволяє скоротити час на розробку, тестування нової функціональності. Дотримання принципу DRY дозволяє досягти високої ефективності при супроводі програмного продукту: внесення змін і тестування значно спрощуються.

Якщо код не дублюється, то для зміни логіки досить внесення виправлень всього в одному місці. Також значно простіше тестувати одну (нехай і більш складну) функцію, а не набір з десятків однотипних. При проходженні DRY спрощується і повторне використання функцій, винесених зі складних алгоритмів, що дозволяє скоротити час розробки і тестування нової функціональності. [5]

На прикладі реалізації програмного забезпечення для моніторингу стану акумулятора портативного комп’ютера (ноутбуку) було розроблено програмне забезпечення, з використанням мови C++ та фреймворку Qt.

2.1 ОПИС АЛГОРИТМУ ФУНКЦІОНУВАННЯ КРОС-ПЛАТФОРМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Сама мова програмування C++ є платформонезалежною мовою, компіляція програм, написаних на ній, здійснюється програмами-компіляторами, які “перекладають” код з більш-менш зрозумілого людині коду (програмного коду) на машинний код. Тобто мова C++ є мовою середнього рівня, а машинний код є інструкціями безпосередньо для “заліза” машини. При цьому компілятори для цієї мови були розроблені під кожну платформу окремо, дотримуючись стандартів програмування на мові C++.

Стандарти мови програмування C++ виходять кожні три роки, які затверджуються Міжнародною організацією стандартів (International Standard Organization, ISO) і на даний час, найновітніший стандарт – це C++20.

При розробці з використанням бібліотеки Qt, для забезпечення кросплатформеності, в даному фреймворку передбачено низку директив компілятору і макросів, в залежності від значень яких, компілятор вирішує, який програмний код варто збирати, а який треба пропустити. Але вставки таких директив не часто доводиться використовувати, так як у бібліотеці Qt передбачено ряд класів, в яких розробники фреймворку передбачили основні кейси для розробки платформозалежного функціоналу. Наприклад, віджет для меню на мобільній операційній системі Android, та на десктопній ОС Windows, виглядає абсолютно по-різному, але код для його ініціалізації один.

2.2 ОБҐРУНТУВАННЯ ВИБОРУ СЕРЕДОВИЩА РОЗРОБКИ

Для розробки даного програмного забезпечення, було обрано середовище для розробки **Qt Creator** (рис 2.2.1), бібліотеку Qt v5.12

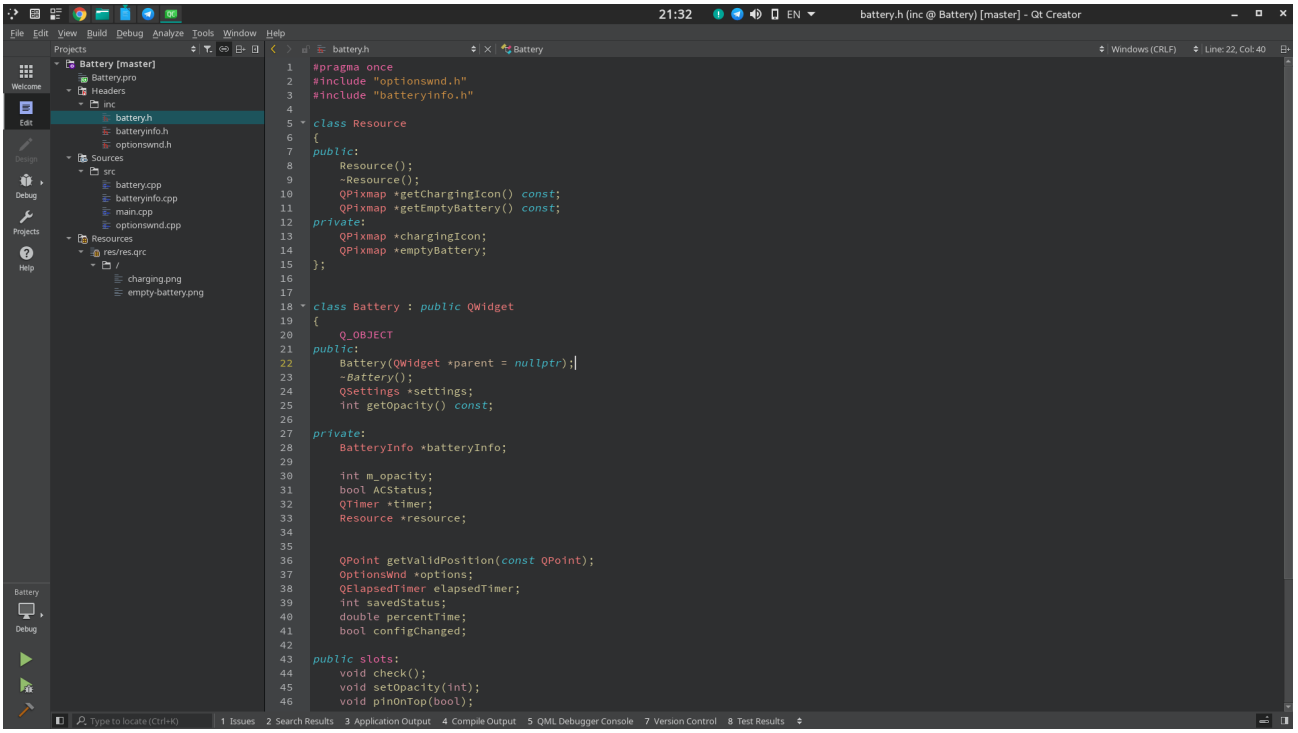

Рисунок 2.2.1 - IDE Qt Creator v4.13

Qt Creator – це інтегроване середовище для розробки, що розроблено компанією Trolltech та призначено для роботи з бібліотекою Qt. Дане середовище створено для того, щоб запропонувати розробникам платформонезалежне середовище для розробки проектів як мові C ++, так і на мові QML.

На будь-якій платформі, що підтримується її розробниками, ви можете використовувати одне і те ж середовище для розробки програмного забезпечення.

Qt Creator включає в себе інтегровану систему допомоги Qt Help, яка надає доступ до документації бібліотеки.

Qt Creator не має влаштованого компілятора, компоновщика і відладчика програмного коду, в ній задіюються доступні на платформі засобі, такі як MinGW для Windows, GNU GCC для Linux,

Також в серидовище вбудовано програму Qt Designer, що дозволяє швидко створювати елементи графічного інтерфейсу користувача, не покидаючи при цьому середовище розробки.

Середовище підтримує такі системи будування проектів, як qmake, cmake та autotools, що дає девелоперам більше можливостей для розробки.

Редактор Qt Creator надає розробникам доволі багато корисних інструментів, що дозволяє значно пришвидшити процес розробки, а саме:

- підсвічування синтаксису (звісно, як же без цього);
- автодоповнення, у тому числі ключових слів із новітніх стандартів мови програмування C++ (C++17, C++20, і т.д.);
- можливість задати стилі вирівнювання блоків, відступів та позицію дужок;
- реалізована можливість працювати с сигнатурами методів, наприклад, автогенерація геттерів та сеттерів для поля класу;
- продвинуті інструменти для рефакторингу коду;
- навігація по коду, через клікання на назву символу з натиснутою клавішею CTRL, для переходу від реалізації до об’явлення;
- можливість викликати допомогу згідно з контекстом програмного коду в будь-якому місці.

3. РОЗРОБКА КРОСПЛАТФОРМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ВІДОБРАЖЕННЯ СТАНУ ЗАРЯДУ АКУМУЛЯТОРУ

Розробка будь-якого проекту, глобального продукту, чи простого додатку для персонального користування, розпочинається однаково - зі створення проекту в обраному середовищі для розробки (IDE), в даному випадку - це Qt Creator.

Будь-який проект, заснований на фреймворку Qt, повинен мати файл конфігурації *.pro, в якому описується, як саме збирати проект, системою для збирання, в даному випадку - qmake. В цьому файлі також декларується, які Qt-модулі необхідні для роботи додатку, назва самого додатку, які файли входять в проект, а саме: файли вихідного коду, файли, які містять ресурси.

Також необхідно пам’ятати, що будь-яка програма на C++, починає свою роботу з так званої “точки входу”, в даному випадку, функції *main(),* яка знаходиться в однойменному файлі main.cpp. Дана функція ініціалізує екземпляр додатку (QApplication [6]) та віджет верхнього рівня - класс, який описує вікно програми.

3.1 КОРОТКИЙ ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Для вирішення завдання в програмі було реалізовано наступні класи:

- Battery (успадковано від QWidget [7]) – безпосередньо саме “вікно” або “віджет верхнього рівня”;
- Resource – клас для того, щоб об’єднати ресурси, що використовуються для відображення віджету;
- BatteryInfo – клас для отримання інформації про стан акумулятора, де саме і реалізується крос-платформний механізм;
- OptionsWnd (успадковано від QWidget) - клас, що відповідає за відображення вікна для конфігурації додатку;

В програмі було задіяно паттерн проектування *RAII* (Resource Acquisition is Initialization) [8].

RAII - це ідіома, що одержала широке поширення завдяки творцеві мови програмування C ++ - Бьярну Страуструпу, і розшифровується як "Resource Acquisition is Initialization" - захоплення ресурсу є ініціалізація.

Ідіома дуже проста і коротко описується в такий спосіб: в конструкторі об'єкт отримує доступ до якогось ресурсу (наприклад, відкривається файл або встановлюється з'єднання з мережі до бази даних) і зберігає хендлер ресурсу в private-полі класу, а при виклику деструктура цей ресурс звільняється (закривається файл або з'єднання до БД). При оголошенні об'єкта даного класу на стеку відбувається і його ініціалізація з викликом конструктора, захоплюючий ресурс. При виході з області видимості об'єкт виштовхується з стека, але перед цим викликається деструктор об'єкта, який і звільняє захоплений ресурс.

Реалізацію даного патерну в програмі можна побачити в файлі *BatteryInfo.cpp* (реалізація класу BatteryInfo, який потрібен для одержання інформації від системи про стан акумулятора)

В конструкторі класу захоплюється ресурс – файл із файлової системи procfs [9] в Linux:

```
BatteryInfo::BatteryInfo()

{

#ifdef Q_OS_LINUX

    QString absolutePath = "/sys/class/power_supply/";

    capacity = nullptr;

    status   = nullptr;

    // Find path of battery in proc filesystem.

    QDirIterator it(absolutePath);

    // Iterate through the directory using the QDirIterator

    while (it.hasNext()) {

        QString filename = it.next();

        QFileInfo file(filename);

        if (!file.isDir()) { // Check if it's a dir

            continue;

        }

        if (file.fileName().contains(QString("BAT"), Qt::CaseInsensitive)) {

            absolutePath += file.fileName();

            capacity = new QFile(absolutePath +  "/capacity");
```

```
            status   = new QFile(absolutePath +  "/status");
            capacity->open(QIODevice::ReadOnly);
            status->open(QIODevice::ReadOnly);
            std::cout << "reading data from " <<
            absolutePath.toStdString() << std::endl;
            break;
        }
    }
    if (!status || !capacity) {
        std::cerr << "no battery connected :(";
    }
#endif
}
```

а в деструкторі ресурси звільняються:

```
BatteryInfo::~BatteryInfo()
{
#ifdef Q_OS_LINUX
    if(status) status->close();
    if (capacity)capacity->close();
    delete status;
    delete capacity;
#endif
}
```

В ОС GNU/Linux для того, щоб отримати інформацію про ACPI, можна зчитати її з директорії **/sys/class/power_supply/** псевдо файлової системи *procfs* [9]. В даній директорії потрібно знайти ту директорію, в якій зберігається інформація про стан акумулятора, найчастіше це BAT0, BAT1 і т.д. Для того, щоб знайти цю директорію, перебираємо всі директорії циклом та знаходимо її по найменуванню:

```
QDirIterator it(absolutePath);
    // Iterate through the directory using the QDirIterator
    while (it.hasNext()) {
        QString filename = it.next();
        QFileInfo file(filename);
        if (!file.isDir()) { // Check if it's a dir
            continue;
        }
       if (file.fileName().contains(QString("BAT"), Qt::CaseInsensitive)) {
            absolutePath += file.fileName();
            capacity = new QFile(absolutePath +  "/capacity");
            status   = new QFile(absolutePath +  "/status");
            capacity->open(QIODevice::ReadOnly);
            status->open(QIODevice::ReadOnly);
            std::cout << "reading data from " << absolutePath.toStdString()
                      << std::endl;
            break;
        }
    }
```

Для ОС Windows достатньо створити структуру SYSTEM_POWER_STATUS: та зчитати інформацію викликом однієї функції з аргументом хендлера цієї структури:

```
...
int BatteryInfo::getCapacity() const
{
#ifdef Q_OS_WIN
    SYSTEM_POWER_STATUS ps;
    GetSystemPowerStatus(&ps);
    return ps.BatteryLifePercent;
#endif

…
```

Структура описана в файлі winbase.h [10] стандартної бібліотеки наступним чином:

```
typedef struct _SYSTEM_POWER_STATUS {
BYTE ACLineStatus;
BYTE BatteryFlag;
BYTE BatteryLifePercent;
BYTE SystemStatusFlag;
DWORD BatteryLifeTime;
DWORD BatteryFullLifeTime;
} SYSTEM_POWER_STATUS, *LPSYSTEM_POWER_STATUS;
```

На даний момент в додатку потрібні такі поля як:

- для отримання статусу акумулятора (чи він підключений, заряджається або розряджається) використовуємо поле `BatteryFlag`;
- `BatteryLifePercent` – для отримання інформації про заряд акумулятора у відсотках.

Сама кросплатформеність в даному випадку реалізовується за допомогою директив компілятору, які вказують, для яких платформ який код використовувати.

Для прикладу, розглянемо уривок програмного коду, а саме реалізацію методу отримання заряду акумулятора:

```
int BatteryInfo::getCapacity() const
{

#ifdef Q_OS_WIN
   SYSTEM_POWER_STATUS ps;
   GetSystemPowerStatus(&ps);
   return ps.BatteryLifePercent;
#endif

#ifdef Q_OS_LINUX

   if (!this->capacity || !this->capacity->isOpen()) {
      std::cerr << "No battery found :(\n";
      return -1;
   }
```

```
    QTextStream in(capacity);
    in.seek(0);
    int percent;
    in >> percent;
    return percent;
#endif
}
```

`#ifdef Q_OS_WIN` [10] – директива компілятору, в залежності від якої, якщо макрос `Q_OS_WIN` визначений (а він визначений тільки на ОС Windows), то весь код нижче буде зібрано і відкомпільовано у executable-файл.

3.2 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Розробка даного програмного забезпечення для операційних систем Windows та Linux відрізняється невеликим обсягом програмного коду. Реалізація основних компонентів додатку, таких як основне вікно, що являється “кастомним” віджетом (з англійської, custom - незвичайний, виготовлений під конкретну конфігурацію), для обох операційних систем не відрізняється.

Відмінності в програмному коді під конкретну платформу є тільки в самих функціях, які отримують дані про заряд аккумулятору портативного пристрою від операційної системи. В даному випадку, в додатку реалізовані такі методи, як: отримання поточного статусу підключення пристрою до мережі (підключений, або не підключений), отримання поточного обсягу батареї у відсотках, та статусу акумулятору, що може бути у наступних станах: розряджається, заряджається, та повністю заряджений.

На обох вищевказаних операційних системах зовнішній вигляд нічим не відрізняється (див. рисунок Б.1-4 Додатку Б.), адже код, відповідальний за малювання зовнішнього вигляду віджету працює однаково. Але можна помітити несуттєві відмінності у відображенні віконця конфігурації конфігурації віджету - елементи управління, такі як слайдери, перемикачі, відображаються згідно з встановленим стилем графічного середовища операційної системи, адже для даних віджетів (елементів) не було задано свій, “кастомний” стиль.

Конфігурація віджету, а саме: прозорість, його позиція, а також параметр “Always on top” (поверх всіх вікон), не скидається при наступному завантаженні, адже даний випадок було передбачено, і всі параметри зберігаються. В програмі забезпечується запис параметрів до системного реєстру у випадку використання ОС Windows, або в директорію з конфігурацією користувача ОС GNU/Linux. Даний механізм також забезпечується влаштованим Qt-классом (QSettings [11]), в якому передбачено зберігання конфігурації додатку для різних операційних систем.

При зміні конфігурації (параметрів) додатку, для уникнення таких помилок користувача, як випадкове переміщення віджету за межі робочого столу (Desktop), або встановлення прозорості на нуль (після чого користувачу буде неможливо побачити віджет та якось з ним взаємодіяти), були встановлені деякі обмеження. Для параметру прозорості дозволено значення в діапазоні від 15 до 100 відсотків. Обмеження щодо позиції віджету розраховуються за допомогою отримання розмірів віджету робочого столу (QDesktopWidget) і при зміні позиції, позиція віджету обмежується розмірами робочого столу.

ВИСНОВКИ

Виходячи з отриманого досвіду, можемо зробити висновки, як саме обрані технології та розроблене програмне забезпечення продемонструвало, яким чином можна високоефективно розробляти додатки на різні платформи, при цьому не змінюючи сам код.

Даний підхід дозволяє з меншими зусиллями робити більш складні додатки, і без зміни програмного коду збирати додатки на різні платформи. Бібліотека Qt була розроблена для досягнення саме для цієї мети.

Даний фреймворк надає розробнику цілий набір інструментів для розробки, та підтримки програмного продукту на різноманітні операційні системи, такі як Windows, Linux, Mac OS X, числі вбудовані (Embedded), а також мобільні операційні системи - iOS, Android, Blackberry та інші.

СПИСОК ЛІТЕРАТУРИ

1. Herbert Schildt. C++: The Complete Reference, 4th Edition
2. wxWidgets. Cross-platform GUI library - www.wxwidgets.org/
3. GTK Library official documentation - https://www.gtk.org/
4. Chromium Embedded Framework wiki documentation - https://bitbucket.org/chromiumembedded/cef/wiki/Home
5. SOLID principles — https://habr.com/ru/post/348286/
6. QApplication Qt class - https://doc.qt.io/qt-5/qapplication.html
7. Qt Documentation — https://doc.qt.io/
8. Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra. Head First Design Patterns 1st edition, 2004 — 242 с.
9. procfs(5) — Linux manual page — https://man7.org/linux/man-pages/man5/procfs.5.html
10. Microsoft WinAPI documentation — https://docs.microsoft.com/en-us/windows/win32/api/
11. QSettings Qt class - https://doc.qt.io/qt-5/qsettings.html

ДОДАТОК А. КОД ПРОГРАМИ

```
# Project created by QtCreator

QT += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = Battery
TEMPLATE = app
DEFINES += QT_DEPRECATED_WARNINGS QT_DLL
CONFIG(debug, release|debug):DEFINES += _DEBUG

INCLUDEPATH += $$PWD/inc

HEADERS += $$PWD/inc/*.h
SOURCES += $$PWD/src/*.cpp

#QMAKE_LFLAGS_RELEASE += -static -static-libgcc
RESOURCES += res/res.qrc


#pragma once
#include "optionswnd.h"
#include "batteryinfo.h"

class Resource
{
public:
    Resource();
    ~Resource();
    QPixmap *getChargingIcon() const;
    QPixmap *getEmptyBattery() const;
private:
    QPixmap *chargingIcon;
    QPixmap *emptyBattery;
};


class Battery : public QWidget
{
    Q_OBJECT
public:
    Battery(QWidget *parent = nullptr);
    ~Battery();
    QSettings *settings;
    int getOpacity() const;

private:
    BatteryInfo *batteryInfo;

    int m_opacity;
    bool ACStatus;
    QTimer *timer;
    Resource *resource;


    QPoint getValidPosition(const QPoint);
    OptionsWnd *options;
    QElapsedTimer elapsedTimer;
    int savedStatus;
    double percentTime;
```

```
    bool configChanged;

public slots:
    void check();
    void setOpacity(int);
    void pinOnTop(bool);

protected:
    void paintEvent(QPaintEvent *);
    void wheelEvent(QWheelEvent *);
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *event);
    void keyPressEvent(QKeyEvent *);
    void mouseDoubleClickEvent(QMouseEvent *event);
    QPoint clickCoords; // mouseclick coords
};
#pragma once
#include <QApplication>
#include <QFile>

class BatteryInfo
{
public:

    bool isConnected() const;
    bool isCharging() const;
    bool isDischarging() const;
    int getCapacity() const;

    BatteryInfo();
    ~BatteryInfo();

#ifdef Q_OS_LINUX
    const QString readStatus() const;

private:
    QFile *capacity;
    QFile *status;
#endif
};

#pragma once

#include <QtWidgets>
class Battery;

class OptionsWnd : public QWidget
{
    Q_OBJECT
    QVBoxLayout *vbl;
    QCheckBox *cb_stayOnTop;
    QSlider *sl_opacity;
    QLabel *lbl_1;
    Battery *battery;
public:
    explicit OptionsWnd(Battery *parent = nullptr);
    ~OptionsWnd();
};

#include "battery.h"
#include <iostream>
using std::cout;
```

```
Battery::Battery(QWidget *parent)
    : QWidget(parent)
{
    options = nullptr;

    savedStatus = 0;
    percentTime = 0;
    configChanged = false;
    ACStatus = false;
    settings = new QSettings("AlexanderAkhtyrtsev", "Battery Widget");

    resize( settings->value("wsize", QSize(45, 90)).toSize());

    m_opacity = settings->value("opacity", 100).toInt();

    setWindowFlags(Qt::Window|Qt::FramelessWindowHint|Qt::Tool);
    setWindowFlag(Qt::WindowStaysOnTopHint, settings->value("ontop", false).toBool());
    setAttribute(Qt::WA_TranslucentBackground);

    resource = new Resource;
    batteryInfo = new BatteryInfo;

    timer = new QTimer(this);
    timer->setInterval(2000);
    QObject::connect(timer, SIGNAL(timeout()), this, SLOT(check()));
    timer->start();
       move(settings->value("wpos",  QPoint(QApplication::desktop()->width()  -  width()  -  10,
40)).toPoint());
}

Battery::~Battery()
{
    delete options;
    delete resource;
    delete settings;
    delete batteryInfo;
    delete timer;
}

int Battery::getOpacity() const
{
    return m_opacity;
}

QPoint Battery::getValidPosition(const QPoint point)
{
    QPoint valid(point);
    QDesktopWidget *desktop = QApplication::desktop();
    int w = desktop->width() - width(),
        h = desktop->height() - height();

    valid.setX(qMax(0, qMin(valid.x(), w)));
    valid.setY(qMax(0, qMin(valid.y(), h)));
    return valid;
}

void Battery::check()
{
    const auto currentCapacity = this->batteryInfo->getCapacity();
    const int delta = savedStatus - currentCapacity;

    // hide widget when battery disconnected
    if (batteryInfo->isConnected()) {
```

```
        if (isHidden()) {
            show();
        }

        repaint();
    }
    else if (!isHidden()) {
        hide();
        std::cout << "widget was hidden cause battery is disconnected.\n";
    }

    // Check if AC status changed
    if (batteryInfo->isCharging() != ACStatus) {
        elapsedTimer.restart();
        ACStatus = batteryInfo->isCharging();
        savedStatus = 0;
        percentTime = 0;
    }


    if (batteryInfo->isDischarging()) {

        // Capacity undefined
        if (!savedStatus) {
            savedStatus = currentCapacity;
        }

        // Capacity changed
        else if (savedStatus != currentCapacity)  {
            percentTime = elapsedTimer.elapsed() / 1000.0 / delta;
            savedStatus = currentCapacity;
        }

    } else if (batteryInfo->isCharging()) {
        if (!savedStatus){
            savedStatus = currentCapacity;
        } else {
            if (savedStatus != currentCapacity) {
                        percentTime  =  elapsedTimer.elapsed()  /  1000.0  /  (currentCapacity  -
savedStatus);
                savedStatus = currentCapacity;
            }
        }
    } else {
        // if status is unknown or battery is full
        savedStatus = 0;
    }



    if (configChanged) {
        settings->setValue("opacity", m_opacity);
                    settings->setValue("ontop",     static_cast<bool>(this->windowFlags()   &
Qt::WindowStaysOnTopHint));
        settings->setValue("wpos", this->pos());
        settings->setValue("wsize", this->size());
        configChanged = false;
    }

}

void Battery::setOpacity(int opacity)
{
```

```
    m_opacity = opacity;
    configChanged = true;
    this->repaint();
}

void Battery::pinOnTop(bool b)
{
    setWindowFlag(Qt::WindowStaysOnTopHint, b);
    configChanged = true;
}

void Battery::paintEvent(QPaintEvent *)
{
    const auto capacity = batteryInfo->getCapacity();
    QBrush brush( capacity < 21 ? Qt::red : capacity < 40 ? Qt::yellow : Qt::green );
    QRect r = rect();

    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing, true);
    painter.setOpacity((qreal)(m_opacity)/100);

    // drawing empty battery
    painter.setBrush( QBrush(resource->getEmptyBattery()->scaled(r.width(), r.height())) );
    painter.setPen(Qt::NoPen);
    painter.drawRect(r);

    // Drawing status
    painter.setCompositionMode(QPainter::CompositionMode_SourceAtop);
    painter.setBrush(brush);
    r.adjust(0, r.height() * (100 - capacity)  / 100, 0, 0);
    painter.drawRect(r);


    // Charging icon
    if ( batteryInfo->isCharging() ) {
        painter.save();
        r = rect();
        r = QRect(r.width()*0.25, r.height() * 0.25, r.width() * 0.5, r.height()*0.5);
        painter.setOpacity(painter.opacity() * 0.5);
        painter.drawPixmap(r, *resource->getChargingIcon());
        painter.restore();
    }

    QFont fnt = painter.font();
    fnt.setPixelSize(rect().height() * 0.15);
    QFontMetrics fmt(fnt);

    QString percentage = QString::number(capacity) + "%";

    auto getTimeStr = [](const QString &title, const QTime &time){
         return QString("\n" + title + ": " + (time.hour() ? time.toString("h") + "h " : "") +
time.toString("m") + "m");
    };

    QString onBatteryStr = "",
            timeleft     = "",
            timeUntilful = "";

    if (this->batteryInfo->isDischarging()) {
        QTime time(0,0,0); time = time.addSecs(elapsedTimer.elapsed() / 1000);
        onBatteryStr = getTimeStr("On battery", time);

        if (percentTime) {
```

```
            int sec = percentTime * capacity;
            QTime time(0,0,0); time = time.addSecs(sec);
            timeleft = getTimeStr("Time left", time);
        }

    } else if (batteryInfo->isCharging() && percentTime) {
        QTime time(0,0,0); time = time.addSecs(percentTime * (100 - capacity));
        timeUntilful = getTimeStr("Until Full", time);
    }

              this->setToolTip(QString(percentage    +    timeUntilful    +    onBatteryStr    +
timeleft).replace(QRegExp("[\\s]0[\\s][hms]"), ""));
    int fw = fmt.horizontalAdvance(percentage);
    painter.setFont(fnt);
    painter.setPen(QPen(Qt::black));
     painter.drawText(QRect(rect().width() / 2 - fw/2, rect().height()/2 - fmt.height()/2, fw,
fmt.height()), percentage);

    painter.end();
}

void Battery::wheelEvent(QWheelEvent *pe)
{
    int d = pe->angleDelta().y() > 0 ? 10 : -10;
    this->resize( qMax(35, qMin(width()+d/2, 300)), qMax(70, qMin(height()+d, 600)) );

    // fix position, dont let move over desktop
    this->move(this->getValidPosition(pos()));
    configChanged = true;
}


void Battery::mousePressEvent(QMouseEvent *event)
{
    clickCoords = event->pos();
}

void Battery::mouseMoveEvent(QMouseEvent *event)
{
    QPoint pos( event->globalX() - clickCoords.x(), event->globalY() - clickCoords.y());
    this->move( this->getValidPosition(pos) );
    configChanged = true;
}

void Battery::keyPressEvent(QKeyEvent *pe)
{
    switch(pe->key()) {
        case Qt::Key_Escape:
            timer->stop();
            QApplication::exit(0);
    }
}

void Battery::mouseDoubleClickEvent(QMouseEvent *)
{
    if (!options) {
        options = new OptionsWnd(this);
    }

    options->show();
    options->move(this->pos());
}
```

```
Resource::Resource()
{
    chargingIcon = new QPixmap(":/charging.png");
    emptyBattery = new QPixmap(":/empty-battery.png");
}

Resource::~Resource()
{
    delete chargingIcon;
    delete emptyBattery;
}

QPixmap *Resource::getChargingIcon() const
{
    return chargingIcon;
}

QPixmap *Resource::getEmptyBattery() const
{
    return emptyBattery;
}
#include "batteryinfo.h"

#ifdef Q_OS_LINUX
#include <QFileInfo>
#include <QDir>

#include <iostream>
#include <QDebug>
#include <QDirIterator>
#include <QFile>
#include <QTextStream>
#endif



BatteryInfo::BatteryInfo()
{

#ifdef Q_OS_LINUX
    QString absolutePath = "/sys/class/power_supply/";

    capacity = nullptr;
    status   = nullptr;


    // Find path of battery in proc filesystem.

    QDirIterator it(absolutePath);

    // Iterate through the directory using the QDirIterator
    while (it.hasNext()) {
        QString filename = it.next();
        QFileInfo file(filename);

        if (!file.isDir()) { // Check if it's a dir
            continue;
        }

        if (file.fileName().contains(QString("BAT"), Qt::CaseInsensitive)) {
            absolutePath += file.fileName();
```

```
            capacity = new QFile(absolutePath +  "/capacity");
            status   = new QFile(absolutePath +  "/status");

            capacity->open(QIODevice::ReadOnly);
            status->open(QIODevice::ReadOnly);

            std::cout << "reading data from " << absolutePath.toStdString() << std::endl;
            break;
        }
    }

    if (!status || !capacity) {
        std::cerr << "no battery connected :(";
    }
#endif
}


BatteryInfo::~BatteryInfo()
{
#ifdef Q_OS_LINUX
    if(status) status->close();
    if (capacity)capacity->close();
    delete status;
    delete capacity;
#endif
}



int BatteryInfo::getCapacity() const
{
#ifdef Q_OS_WIN
    SYSTEM_POWER_STATUS ps;
    GetSystemPowerStatus(&ps);
    return ps.BatteryLifePercent;
#endif

#ifdef Q_OS_LINUX

    if (!this->capacity || !this->capacity->isOpen()) {
        std::cerr << "No battery found :(\n";
        return -1;
    }

    QTextStream in(capacity);
    in.seek(0);
    int percent;
    in >> percent;
    return percent;
#endif
}

#ifdef Q_OS_LINUX
const QString BatteryInfo::readStatus() const
{
    QString currentStatus = "";

    if (this->status && this->status->isOpen()) {
        QTextStream in(this->status);
        in.seek(0);
        in >> currentStatus;
    }
```

```
    return currentStatus;
}

#endif
bool BatteryInfo::isConnected() const
{
#ifdef Q_OS_WIN
    SYSTEM_POWER_STATUS ps;
    GetSystemPowerStatus(&ps);
    return ps.BatteryFlag < 128;
#endif

#ifdef Q_OS_LINUX
    return this->status && this->status->isOpen();
#endif
}

bool BatteryInfo::isCharging() const
{
#ifdef Q_OS_WIN
    SYSTEM_POWER_STATUS ps;
    GetSystemPowerStatus(&ps);
    return ps.BatteryFlag & 8;
#endif

#ifdef Q_OS_LINUX
    return this->readStatus() == "Charging";
#endif
}

bool BatteryInfo::isDischarging() const
{
#ifdef Q_OS_WIN
    SYSTEM_POWER_STATUS ps;
    GetSystemPowerStatus(&ps);
    return (ps.BatteryFlag != 255 && ps.BatteryFlag != 128 && !(ps.BatteryFlag & 8));
#endif

#ifdef Q_OS_LINUX
    return this->readStatus() == "Discharging";
#endif
}
#include "battery.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Battery w;
    w.check();
    return a.exec();
}
#include "battery.h"

OptionsWnd::OptionsWnd(Battery *parent) : QWidget(parent)
{
    setWindowFlags(Qt::Tool);

    setWindowTitle("Options");
    battery = parent;
    cb_stayOnTop = new QCheckBox("Always on top");
    cb_stayOnTop->setChecked(battery->settings->value("ontop", false).toBool());
    QObject::connect(cb_stayOnTop, SIGNAL(clicked(bool)), battery, SLOT(pinOnTop(bool)));
```

```
    lbl_1 = new QLabel("Opacity");

    sl_opacity = new QSlider(Qt::Horizontal);
    sl_opacity->setRange(15, 100);
    sl_opacity->setValue(battery->getOpacity());
    QObject::connect(sl_opacity, SIGNAL(valueChanged(int)), battery, SLOT(setOpacity(int)));

    vbl = new QVBoxLayout;
    vbl->addWidget(cb_stayOnTop);
    vbl->addWidget(lbl_1);
    vbl->addWidget(sl_opacity);
    setLayout(vbl);
    setFixedSize(180, 90);
}

OptionsWnd::~OptionsWnd()
{
    delete vbl;
    delete lbl_1;
    delete sl_opacity;
    delete cb_stayOnTop;
}
```

ДОДАТОК Б. ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ

Віконце для налаштування відображення (рис. 1) включає в себе такі параметри:

Always on top – закріплення поверх всіх вікон;

Opacity – прозорість віджету.

При заряджанні відображається відповідний індикатор:

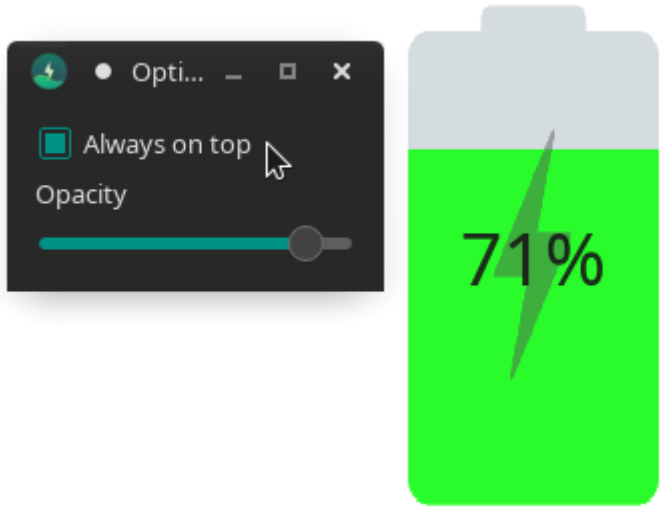

Рисунок Б.1. - Робота програми при високому заряді (ОС Linux)

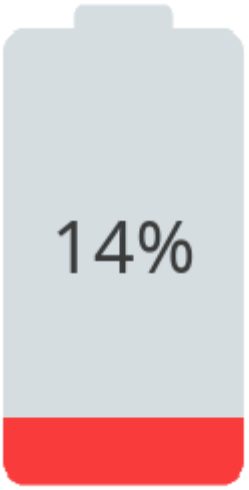

Рисунок Б.2. – Відображення віджету при низькому рівні заряді

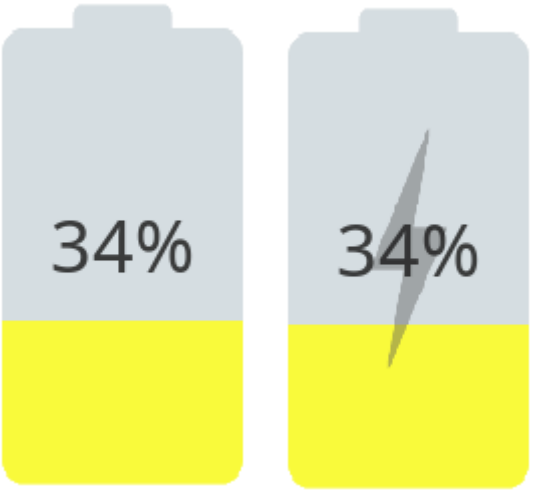

Рисунок Б.3. – Відображення віджету при середньому рівні заряду

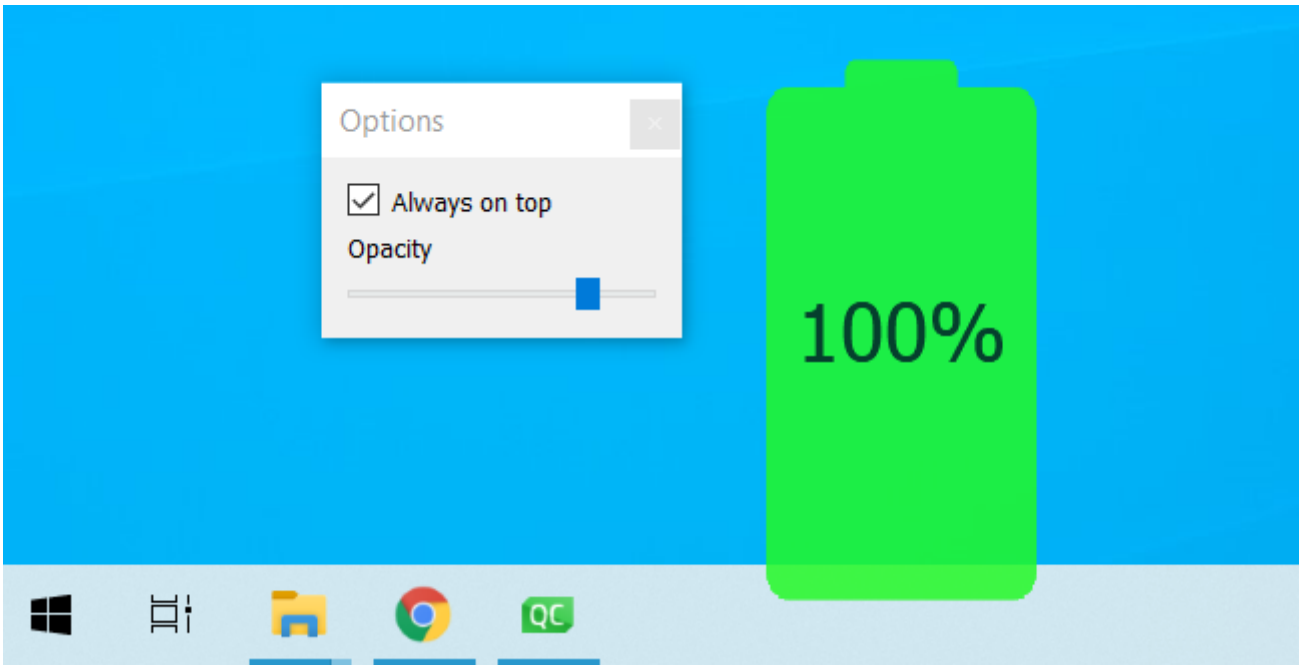

Рисунок Б.4. - Робота додатку на операційній системі Windows 10