Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Менеджер работы батареии на ОС WINDOWS

БГУИР КП I – 40 04 01

|  |  |  |
| --- | --- | --- |
| Выполнил  студент гр. 053505 |  | О. Э.  Осадчий |
| Проверила: |  | А. А. Калиновская |

Минск 2022

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 3](#_Toc121361058)

[ВВЕДЕНИЕ 4](#_Toc121361059)

[1 Платорма программного обеспечения 5](#_Toc121361060)

[1.1 Qt 5](#_Toc121361061)

[1.2 Операционная система Windows 7](#_Toc121361062)

[2 Теоретическое обоснование разработки программного продукта 9](#_Toc121361063)

[3 Проектирование функциональных возможностей программы 16](#_Toc121361064)

[3.1 Функциональные возможности программы 16](#_Toc121361065)

[3.2 Программная реализация 23](#_Toc121361066)

[4 Архитектура разрабатываемой программы 32](#_Toc121361067)

[ЗАКЛЮЧЕНИЕ 36](#_Toc121361068)

[СПИСОК ЛИТЕРАТУРЫ 37](#_Toc121361069)

[ПРИЛОЖЕНИЕ А 38](#_Toc121361070)

[ПРИЛОЖЕНИЕ Б 52](#_Toc121361071)

[ПРИЛОЖЕНИЕ В 53](#_Toc121361072)

[ПРИЛОЖЕНИЕ Г 54](#_Toc121361073)

# ВВЕДЕНИЕ

В современном мире каждый человек пользуется ноутбуком. Мы не можем представить свою жизнь без него. Ноутбук нужен нам для большого количества различных задач. Ноутбуки используются везде: при работе с бортовыми компьютерами машин, отладке выпускаемой продукции различных заводов, для написания программных продуктов. Человечество использует ноутбуки повсеместно, и для нас очень важно, чтобы данные агрегаты работали исправно. А одной из его самых важных частей, наряду с процессором, материнской платой, видеокартой, является батарея.

Батарея является источником питания ноутбука. И для нас, как для пользователя, критически важно наблюдать за состоянием батареи. Ведь, если мы не будем следить за этим, то в какой-то момент можем обнаружить, что наш ноутбук не включается, или включается, но не может работать без подключения к сети питания, или же, банально, держит заряд аккумулятора критически мало. Собственно говоря, для того, чтобы пользователь не сталкивался с такими проблемами, ему необходимо специальное программное обеспечение. Например, менеджер работы батареи. Такой менеджер может показывать текущий заряд аккумулятора. Но, что важнее, такой менеджер сможет показать количество циклов зарядки-разрядки аккумулятора, текущую ёмкость аккумулятора, тип аккумулятора, что поможет, к примеру, не использовать ноутбук с аккумулятором определённого типа в неподходящих условиях, либо же знать о проблемах своего типа аккумулятора и приступить к замене.

Резюмируя всё вышеизложенное можно сказать, что аккумулятор – это из важнейших частей компьютера, за которой, как и за процессором, видеокартой и прочим оборудованием требуется уход и наблюдение. Существует огромное количество приложений отслеживания характеристик процессора и видеокарты, а также управления ими, однако же аналогичных приложений для батареи почти нет. Из чего вытекает актуальность данной темы.

# Платорма программного обеспечения

## 1.1 Qt

Qt — это кроссплатформенный фреймворк для разработки ПО на языке программирования C++ (и не только). Также имеется и для Ruby — QtRuby, для Python — PyQt, PHP — PHP-Qt и других языков программирования. Разрабатывается компанией Trolltech с 1996 года.

С использованием этого фреймворка написано множество популярных программ: 2ГИС для Android, Kaspersky Internet Security, Virtual Box, Skype, VLC Media Player, Opera и другие. KDE — это одно из окружений рабочего стола со множеством программ для Linux написано с использованием фреймворка Qt.

Qt полностью объектно-ориентированная, кроссплатформенная. Дает возможность разрабатывать платформо-независимое ПО, написанный код можно компилировать для Linux, Windows, Mac OS X и других операционных систем. Включает в себя множество классов для работы с сетью, базами данных, классы-контейнеры, а также для создания графического интерфейса и множество других (чуть ниже).

Qt использует MOC (Meta Object Compiler) для предварительной компиляции программ. Исходный текст программы обрабатывается MOC, который ищет в классах программы макрос Q\_OBJECT и переводит исходный код в мета-объектный код, после чего мета-объектный код компилируется компилятором C++. MOC расширяет функциональность фреймворка, благодаря ему добавляются такие понятия, как слоты и сигналы.

В Qt имеется огромный набор виджетов (Widget), таких как: кнопки, прогресс бары, переключатели, checkbox, и другие — они обеспечивают стандартную функциональность GUI (графический интерфейс пользователя). Позволяет использовать весь функционал пользовательского интерфейса — меню, контекстные меню, drag&drop.

**Qt** имеет среду разработки **Qt Creator**. Она включает в себя **Qt Designer**, с помощью которого можно создавать графический интерфейс. Визуальное создание интерфейса позволяет легко и просто создавать интерфейс, перетаскивая различные виджеты(выпадающие списки, кнопки, переключатели) на форму.

Qt поставляется вместе с Qt Assistant — это огромный интерактивный справочник, содержащий в себе информацию по работе с Qt. К сожалению полностью не переведен на русский. В состав Qt также входит Qt Linguist, которая позволяет локализировать приложение для разных языков.

Состав библиотеки Qt.

Библиотека Qt состоит из различных модулей, которые подключаются при помощи директивы #include. В состав входят:

1. QtCore — классы ядра библиотеки Qt, они используются другими модулями.
2. QtGui — модуль содержит компоненты графического интерфейса.
3. QtNetwork — модуль содержит классы для работы с сетью. В него входят классы для работы с протоколами FTP, HTPP, IP и другими.
4. QtOpenGL — модуль содержит классы для работы с OpenGL.
5. QtSql — содержит классы для работы с различными базами данных с использованием языка SQL.
6. QtSvg — содержит классы, позволяющие работать с данными Scalable Vector Graphics (SVG).
7. QtXml — классы для работы с XML.
8. QtScript — классы для работы с Qt Scripts.

Имеются и другие модули.

В данный момент Qt распрастраняется по 3-м лицензиям: Qt Commercial(собственическая), GNU GPL, GNU LGPL.

В настоящее время Qt фреймворк активно развивается. Имеет интуитивно понятное API, огромную документацию с большим количеством примеров, мощнейшую среду разработки QtCreator, а также дополнительный инструментарий.

Также важной часть работы в данном проекте с Qt – выбор компилятора для проекта. Передо мной стоял выбор между Desktop Qt 6.3.2 MinGW 64-bit, и Desktop Qt 6.3.2 MSVC2019 64-bit. Предпочтение было отдано в пользу последнего, поскольку во время практических попыток запуска проекта с компилятором MinGW, который я хотел использовать для последующей кроссплатформенной разработки, передо мной встала проблема подключения необходимых библиотек, которые невозможно подключить с этим компилятором.

## 1.2 Операционная система Windows

Microsoft Windows представляет целую серию операционных систем и рабочих сред, разработанных корпорацией Microsoft. Самая первая версия их операционных систем – это MS-DOS. Она была представлена в 1985 году и имела графический пользовательский интерфейс. В ней была обеспеченна поддержка нескольких документов, поддержка компьютерной мыши, выпадающее меню, и всё это было возможным увидеть в цветах.

Дальнейшим развитием, а точнее, заметой MS-DOS стали современные Windows. Само название Windows, на русском языке означает "Окна" и имеет в нашем языке синонимы Виндовс, Вундоуз и другие производные полученные после перевода.

Следующие версии Windows постепенно заменяли многое построенное в MS-DOS, аппаратный и программный функционал. Все внесённые изменения Microsoft в MS-DOS с дальнейшей интеграцией поспособствовали получению развитой операционной системы.

В настоящее время Microsoft Windows является самой популярной ОС, не только из-за привычного удобства для использования и внушительного функционала, но и из-за высокого уровня интеграции с возможностями её ядра и другого программного обеспечения, в том числе MS Office.

Хотя по началу такая программная интеграция имела проблемы с безопасностью, сегодня же Microsoft Windows широко используется не только на персональных компьютерах, но и на серверных версиях вычислительной техники.

В качестве комплексной операционной системы, все версии Microsoft Windows поставляются с предустановленным программным обеспечением, которое можно использовать сразу после её установки. Основные текстовые редакторы и калькулятор стали доступными для использования в самой первой версии Windows. Windows 98 привнесла Media Payer, Internet Explorer и Outlook Express. Начиная с Windows Vista появились фотогалерея DVD Maker и боковая панель, которая в Windows 7, 8, 10 представлена гаджетами-информерами. Начиная со второго пакета обновления Windows XP и с выходом Windows Vista появилась встроенная функция безопасности – фаервол (брандмауэр). Эта функциональность у современных ОС Windows реализуется на уровне ядра. Благодаря песочнице, вредоносное поведение не может вызвать общесистемную нестабильность или нарушение безопасности, всё ПО работает за приделами ядра системы, включая сторонние антивирусные системы.

В дополнение к общим пользовательским версиям операционных систем Windows для ПК, Microsoft выпустила серверные версии, мобильные и встроенные версии своей операционной системы. Система Windows Server имеет конкурентный административный пакет, включает в себя услуги по обновлению, сервер хранения, веб-сервер, медиа сервер и ещё более внушительную функциональность чем может показаться новичку.

Это пакет предназначен в первую очередь для корпоративного использования, организациям которым необходимо администрирование локальной сети и интеграция в неё различных типов компьютеров в своих целях.

С каждым днём Microsoft Windows выпускает нововведения и обновления безопасности. Многим пользователи вполне устраивает возможность работы в этом логически построенном семействе операционных систем Windows.

На данный момент рядовому пользователю доступна самая последняя версия Windows – Windows 11. Однако самой популярной и наиболее используемой версией является Windows 10, под которую всё ещё выходят обновления.

# Теоретическое обоснование разработки программного продукта

Актуальность данной темы вытекает из рынка программных продуктов в данном сегменте. А именно: во-первых, сама операционная система Windows даёт пользователю небольшое количество информации о батарее. В меню настроек батареи всё, что может увидеть пользователь – это текущий заряд, статус зарядки устройства и время до полной зарядки/разрядки, а также использование батарее различными приложениями за последние 24 часа, либо за прошедшую неделю. Системные параметры аккумулятора в Windows представлены на рисунках 6 и 7.

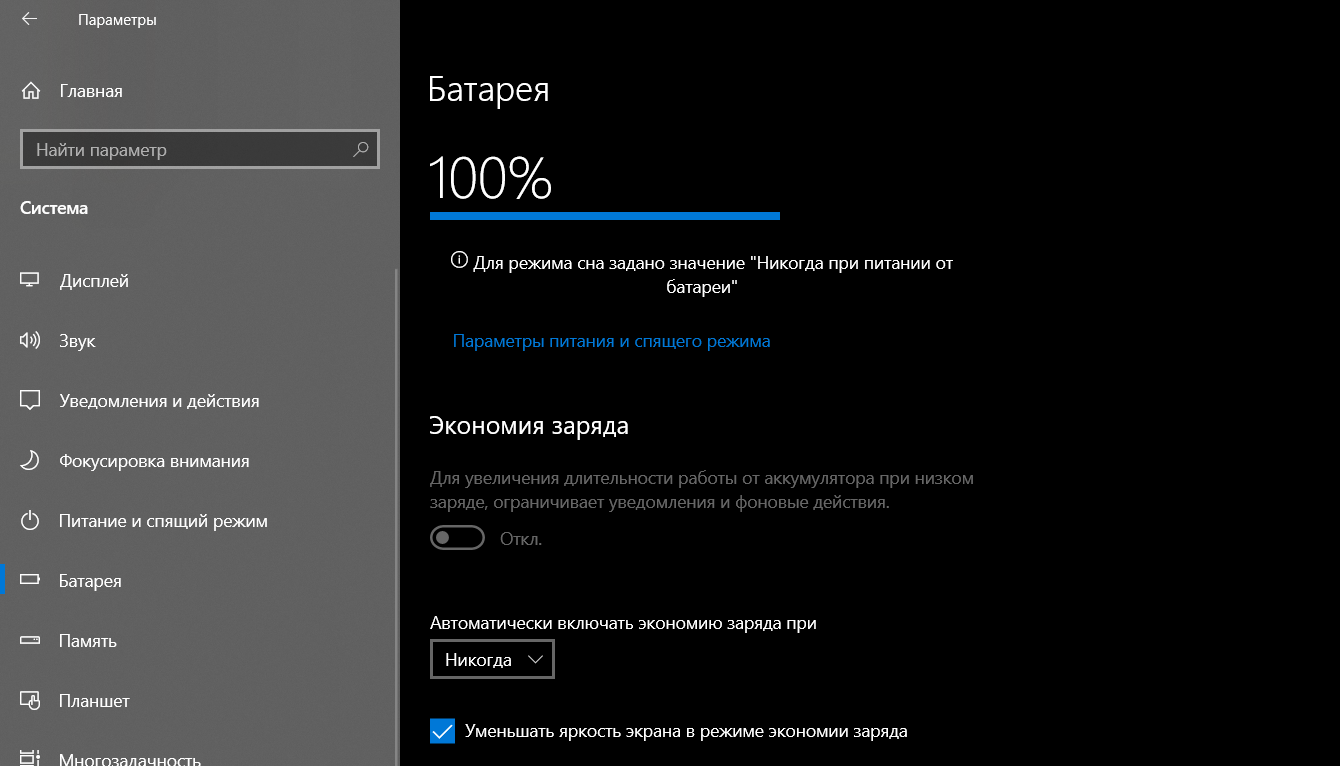


Рисунок 6 - Системные параметры аккумулятора Windows

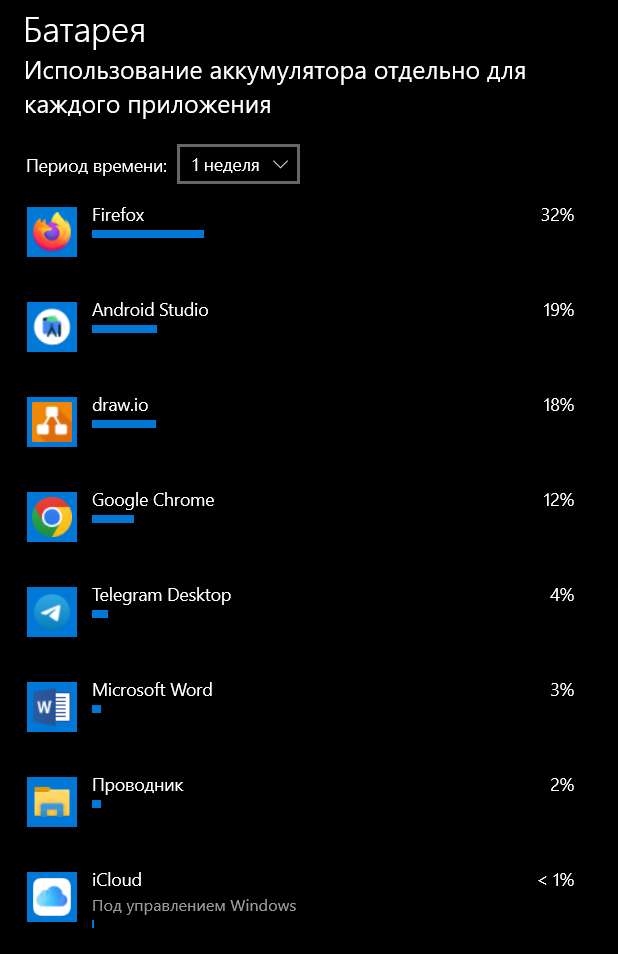


Рисунок 7 - Системный подсчёт использования батареи

В соседнем же меню «питание и спящий режим» пользователь имеет возможность установить значение времени «простоя» после которого компьютер или ноутбук будет переведён в режим сна. Меню показано на рисунке 8.

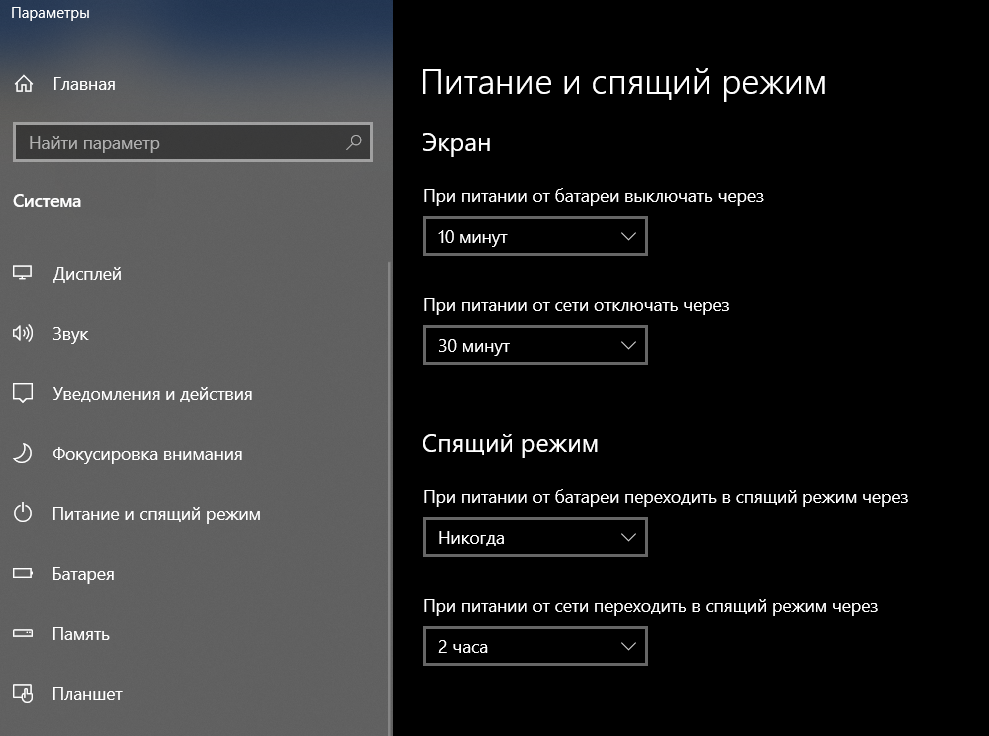


Рисунок 8 - Питание и спящий режим

Во-вторых, как описывалось выше (во введении), такой параметр, как тип аккумулятора в современном мире играет значительную роль при покупке ноутбука.

В основе производства большинства аккумуляторных батарей для ноутбуков находятся нескольких типов аккумуляторных элементов:

1 NiCad (Никель-кадмиевые), изображённый на рисунке 1.



Рисунок 1 - Никель-кадмиевая батарея ноутбука.

2 NiMh (никель-металлогидридные), изображённый на рисунке 2 и 3.



Рисунок 2 - Никель-металлогидридная батарея ноутбука



Рисунок 3 - Никель-металлогидридная батарея ноутбука изнутри  
3 LiIon (литий-ионные), изображённый на рисунке 4.



Рисунок 4 - Литий-ионный аккумулятор ноутбука

4 LiPoly (литий-полимерные), изображённый на рисунке 5.



Рисунок 5 - Литий-полимерный аккумулятор ноутбука

Каждый тип характеризуется определёнными особенностями:

Аккумуляторы на основе элементов NiMh имеют ёмкость почти в два раза больше, чем аккумуляторы на основе NiCad, при одинаковом размере и весе. Также никель-металлогидридные аккумуляторы более экологически безопасные, чем никель-кадмиевые, из-за отсутствия в них тяжёлых металлов. И, наконец, NiMh аккумуляторы менее подвержены «эффекту памяти».

Литий-ионные аккумуляторы (LiIon) при весе на 30% меньшем, чем никель-металлогидридные обладают вдвое большей ёмкостью и абсолютно не подвержены «эффекту памяти». Основными их недостатками можно назвать относительно небольшой диапазон рабочих температур, а также стоимость их выше, чем стоимость NiMh аккумуляторов. Но, не смотря на это, именно LiIon аккумуляторы в настоящее время в основном используются в мобильных устройствах, таких как ноутбуки, мобильные телефоны и т.д.

Следующим этапом эволюции аккумуляторных батарей стала технология LiPoly. В аккумуляторах на основе литий-полимерных элементов нет жидкого электролита, что делает их более безопасными для пользователя ноутбука, мобильного телефона, чем их предшественники. Также они значительно легче, имеют больший срок службы и более широкий диапазон рабочих температур.

«Эффект памяти» - побочный эффект, который возникает из-за химических реакций, происходящих внутри аккумуляторных элементов. Эта проблема характерна только для никель-металлогидридных и никель-кадмиевых аккумуляторов.

Суть этого эффекта заключается в том, что аккумуляторная батарея ноутбука «запоминает» количество энергии, отданное в последнем цикле разряда, и уравнивает свою ёмкость с объёмом отданной энергии. В результате этого при следующем заряде она накопит уже меньше энергии, а со временем её ёмкость может сократиться даже в несколько раз.

Для профилактики возникновения «эффекта памяти» в аккумуляторной батарее Вашего ноутбука необходимо полностью разряжать и заряжать её.

В наше время «эффект памяти» становится пережитком прошлого, т.к. в современных ноутбуках в основном используются аккумуляторные батареи на основе литий-ионных элементов, а в них этот эффект отсутствует.

В-третьих, на данный момент приложений, предоставляющих возможность просмотра аппаратных характеристик батареи, установленной в персональном компьютере (PC), в том числе и в ноутбуках, очень мало. А значит, появляется возможность для создания качественного программного продукта, который будет предоставлять данные возможности, к тому же, на русском языке.

# Проектирование функциональных возможностей программы

## 3.1 Функциональные возможности программы

Проектирование функциональных возможностей программы исходит в первую очередь из возможностей, которые предоставляет операционная система, получить информацию о батарее, а также управлять ей.

Операционная система Windows на уровне пользователя, как я показывал выше, не даёт нам большого количества данных, а управление сводится к включению/выключению режима энергосбережения. На уровне же программиста она даёт больше, однако всё равно довольно сильно ограничивает в возможностях.

Сбор данных о батарее возможен благодаря следующим структурам (объект c++):

1. **SYSTEM\_POWER\_STATUS structure**

**typedef struct** \_SYSTEM\_POWER\_STATUS {

BYTE ACLineStatus;

BYTE BatteryFlag;

BYTE BatteryLifePercent;

BYTE SystemStatusFlag;

DWORD BatteryLifeTime;

DWORD BatteryFullLifeTime;

} SYSTEM\_POWER\_STATUS, \*LPSYSTEM\_POWER\_STATUS;

Составные части данной структуры:

1. BYTE ACLineStatus – состояние питания переменного тока. Принимает значение 1, если батарея подключена к сети питания, 0, если не подключена, 255 при неизвестном статусе.
2. BYTE BatteryFlag – статус зарядки батареи. Принимает значение 1, если заряд более 66%, 2, при заряде мене 33%, 4 при критичном значении заряда аккумулятора, 8 – статус зарядки устройства, 128 – отсутствие батареи в устройстве, 255 – неизвестный статус. Также значение может быть нулём при отсутствии зарядки от сети и нахождением заряда батареи в пределах от нижнего уровня до верхнего.
3. BYTE BatteryLifePercent – процент оставшегося заряда аккумулятора. Значение от 0 до 100, или же 255, если статус неизвестен.
4. BYTE SystemStatusFlag – статус энергосберегающего режима батареи. 0, если выключен – 1, если включен.
5. DWORD BatteryLifeTime – Количество секунд оставшегося времени автономной работы или -1, если оставшиеся секунды неизвестны или если устройство подключено к сети переменного тока.
6. DWORD BatteryFullLifeTime - Количество секунд автономной работы при полной зарядке или -1, если полный заряд батареи неизвестен или если устройство подключено к сети переменного тока.
7. **SYSTEM\_POWER\_STATE enumeration**

**typedef enum** \_SYSTEM\_POWER\_STATE {

PowerSystemUnspecified = 0,

PowerSystemWorking = 1,

PowerSystemSleeping1 = 2,

PowerSystemSleeping2 = 3,

PowerSystemSleeping3 = 4,

PowerSystemHibernate = 5,

PowerSystemShutdown = 6,

PowerSystemMaximum = 7

} SYSTEM\_POWER\_STATE, \*PSYSTEM\_POWER\_STATE;

Структура, содержащая в себе состояния батареи (сон, гибернация, выключена и т.д.).

1. **SYSTEM\_BATTERY\_STATE structure**

**typedef struct** {

BOOLEAN AcOnLine;

BOOLEAN BatteryPresent;

BOOLEAN Charging;

BOOLEAN Discharging;

BOOLEAN Spare1[3];

BYTE Tag;

DWORD MaxCapacity;

DWORD RemainingCapacity;

DWORD Rate;

DWORD EstimatedTime;

DWORD DefaultAlert1;

DWORD DefaultAlert2;

} SYSTEM\_BATTERY\_STATE, \*PSYSTEM\_BATTERY\_STATE;

Составные части данной структуры:

1. BOOLEAN AcOnLine – Состояние питания переменного тока, если выставлено значение TRUE, системное зарядное устройство в данный момент работает от внешнего источника питания.
2. BOOLEAN BatteryPresent - Если этот элемент имеет значение TRUE, то в системе присутствует по крайней мере одна батарея.
3. BOOLEAN Charging - Если этот элемент имеет значение TRUE, то в данный момент аккумулятор заряжается.
4. BOOLEAN Discharging - Если этот элемент имеет значение TRUE, то в данный момент батарея разряжается.
5. BOOLEAN Spare1 – Зарезервировано.
6. BYTE Tag – Зарезервировано.
7. DWORD MaxCapacity – Текущая максимальная ёмкость батареи( выше неё зарядить батарею нельзя).
8. DWORD RemainingCapacity - Расчетная оставшаяся емкость аккумулятора (тот же заряд аккумулятора, только не в процентах, а в милливольтах-час).
9. DWORD Rate - Текущая скорость разряда аккумулятора, в МВт. Ненулевая положительная скорость указывает на зарядку; отрицательная скорость указывает на разрядку. Некоторые батареи сообщают только о скорости разряда. Это значение следует рассматривать как LONG, поскольку оно может содержать отрицательные значения (с установленным старшим битом).
10. DWORD EstimatedTime - Расчетное время, оставшееся от заряда батареи, в секундах.
11. DWORD DefaultAlert1 - Предположение производителя о мощности в МВт/ч, при которой должно возникать предупреждение о низком заряде батареи. Определения низкого уровня варьируются от производителя к производителю. В общем случае состояние предупреждения будет возникать перед низким состоянием, но вы не должны предполагать, что так будет всегда. Чтобы снизить риск потери данных, это значение обычно используется в качестве параметра по умолчанию для аварийного сигнала о критическом заряде батареи.
12. DWORD DefaultAlert2 - Предположение производителя о мощности в МВт/ч, при которой должно появиться предупреждающее предупреждение о заряде батареи. Определения предупреждения варьируются от производителя к производителю. В общем случае состояние предупреждения будет возникать перед низким состоянием, но вы не должны предполагать, что так будет всегда. Чтобы снизить риск потери данных, это значение обычно используется в качестве параметра по умолчанию для сигнала тревоги о низком заряде батареи.
13. **SYSTEM\_POWER\_LEVEL structure**

**typedef struct** {

BOOLEAN Enable;

BYTE Spare[3];

DWORD BatteryLevel;

POWER\_ACTION\_POLICY PowerPolicy;

SYSTEM\_POWER\_STATE MinSystemState;

} SYSTEM\_POWER\_LEVEL, \*PSYSTEM\_POWER\_LEVEL;

Составные части данной структуры:

1. BOOLEAN Enable - Если этот элемент имеет значение TRUE, сигнал тревоги должен быть активирован, когда батарея разряжается ниже значения, установленного в разделе Уровень заряда батареи.
2. BYTE Spare – Зарезервировано.
3. DWORD BatteryLevel - Емкость аккумулятора для данной политики разряда аккумулятора, выраженная в процентах.
4. POWER\_ACTION\_POLICY PowerPolicy - Структура POWER\_ACTION\_POLICY, которая определяет действие, которое необходимо предпринять для этой политики разряда батареи.
5. SYSTEM\_POWER\_STATE MinSystemState - Минимальное состояние спящего режима системы, в которое необходимо перейти, когда батарея разряжается ниже значения, установленного в разделе Уровень заряда батареи. Этот элемент должен быть одним из значений типа перечисления SYSTEM\_POWER\_STATE.
6. **SYSTEM\_POWER\_POLICY structure**

**typedef struct** \_SYSTEM\_POWER\_POLICY {

DWORD Revision;

POWER\_ACTION\_POLICY PowerButton;

POWER\_ACTION\_POLICY SleepButton;

POWER\_ACTION\_POLICY LidClose;

SYSTEM\_POWER\_STATE LidOpenWake;

DWORD Reserved;

POWER\_ACTION\_POLICY Idle;

DWORD IdleTimeout;

BYTE IdleSensitivity;

BYTE DynamicThrottle;

BYTE Spare2[2];

SYSTEM\_POWER\_STATE MinSleep;

SYSTEM\_POWER\_STATE MaxSleep;

SYSTEM\_POWER\_STATE ReducedLatencySleep;

DWORD WinLogonFlags;

DWORD Spare3;

DWORD DozeS4Timeout;

DWORD BroadcastCapacityResolution;

SYSTEM\_POWER\_LEVEL DischargePolicy[NUM\_DISCHARGE\_POLICIES];

DWORD VideoTimeout;

BOOLEAN VideoDimDisplay;

DWORD VideoReserved[3];

DWORD SpindownTimeout;

BOOLEAN OptimizeForPower;

BYTE FanThrottleTolerance;

BYTE ForcedThrottle;

BYTE MinThrottle;

POWER\_ACTION\_POLICY OverThrottled;

} SYSTEM\_POWER\_POLICY, \*PSYSTEM\_POWER\_POLICY;

Составные части данной структуры:

1. POWER\_ACTION\_POLICY PowerButton - Структура POWER\_ACTION\_POLICY, которая определяет действие кнопки питания системы, которое должно инициироваться при нажатии кнопки питания.
2. POWER\_ACTION\_POLICY SleepButton - Структура POWER\_ACTION\_POLICY, которая определяет действие включения системы, которое должно инициироваться при нажатии кнопки перехода системы в спящий режим.
3. POWER\_ACTION\_POLICY LidClose - Структура POWER\_ACTION\_POLICY, которая определяет действие питания системы, которое должно инициироваться при закрытии ноутбука.
4. SYSTEM\_POWER\_STATE LidOpenWake - Состояние максимальной мощности (наивысшее значение Sx), из которого событие открытия ноутбука должно привести систему в действие. Этот элемент должен быть одним из значений типа перечисления SYSTEM\_POWER\_STATE.
5. SYSTEM\_POWER\_STATE MinSleep - Минимальное состояние спящего режима системы (наименьшее значение Sx), поддерживаемое в настоящее время. Этот элемент должен быть одним из значений типа перечисления SYSTEM\_POWER\_STATE.
6. SYSTEM\_POWER\_STATE MaxSleep - Максимальное поддерживаемое в настоящее время состояние спящего режима системы (наивысшее значение Sx). Этот элемент должен быть одним из значений типа перечисления SYSTEM\_POWER\_STATE;
7. DWORD DozeS4Timeout - Время ожидания между переходом в состояние приостановки и переходом в спящий режим гибернации, в секундах. Нулевое значение указывает на то, что никогда не переходите в спящий режим.
8. DWORD BroadcastCapacityResolution - Разрешение изменения текущей емкости батареи, которое должно привести к уведомлению системы об изменении состояния питания системы.
9. DWORD VideoTimeout - Время до выключения дисплея в секундах;
10. BOOLEAN VideoDimDisplay - Если этот элемент имеет значение TRUE, система включает поддержку затемнения дисплея.
11. **BATTERY\_WMI\_CYCLE\_COUNT structure**

**typedef struct** \_BATTERY\_WMI\_CYCLE\_COUNT {

ULONG Tag;

ULONG CycleCount;

} BATTERY\_WMI\_CYCLE\_COUNT, \*PBATTERY\_WMI\_CYCLE\_COUNT;

Составные части данной структуры:

1. ULONG Tag - Метка, идентифицирующая конкретную батарею.
2. ULONG CycleCount - Количество циклов зарядки/разрядки, пройденных батареей, или ноль, если батарея не поддерживает счетчик циклов.
3. **BATTERY\_INFORMATION structure**

**typedef struct** \_BATTERY\_INFORMATION {

ULONG Capabilities;

UCHAR Technology;

UCHAR Reserved[3];

UCHAR Chemistry[4];

ULONG DesignedCapacity;

ULONG FullChargedCapacity;

ULONG DefaultAlert1;

ULONG DefaultAlert2;

ULONG CriticalBias;

ULONG CycleCount;

} BATTERY\_INFORMATION, \*PBATTERY\_INFORMATION;

Составные части данной структуры:

1 Указывает возможности батареи в виде значения ULONG, закодированного одним или несколькими из следующих флагов:

- BATTERY\_SYSTEM\_BATTERY - Устанавливается этот флаг, если батарея может обеспечить общее питание для запуска системы.

- BATTERY\_CAPACITY\_RELATIVE - Устанавливается этот флаг, если драйвер миникласса будет сообщать о емкости батареи и скорости зарядки в процентах от общей емкости и скорости зарядки, а не в абсолютных значениях. В противном случае драйвер мини-класса должен сообщать о мощности в милливатт-часах и скорости в милливаттах.

- BATTERY\_IS\_SHORT\_TERM - Устанавливается этот флаг, если аккумулятор является ИБП, предназначенным для кратковременного безотказного использования. Снимается флаг для любого другого типа устройства.

- BATTERY\_SET\_CHARGE\_SUPPORTED - Установливается этот флаг, если драйвер мини-класс поддерживает настройку заряда батареи в вызовах информации о мини-наборе батареи.

- BATTERY\_SET\_DISCHARGE\_SUPPORTED - Устанавливается этот флаг, если драйвер мини-класс поддерживает настройку разряда батареи в вызовах информации о мини-наборе батареи.

2 Technology - Устанавливается ноль для первичной, не перезаряжаемой батареи или единица для вторичной, перезаряжаемой батареи.

3 DesignedCapacity - Указывает теоретическую емкость новой батареи в милливатт-часах. Если задано значение BATTERY\_CAPACITY\_RELATIVE, единицы измерения не определены.

4 CycleCount - Указывает количество циклов зарядки/разрядки, пройденных батареей, или ноль, если батарея не поддерживает счетчик циклов.

Таким образом, исходя из возможностей доступа к батарее, предоставляемого нам операционной системой, можно составить список функций, которая должна выполнять программа:

1. Выводить текущий заряд батареи.
2. Выводить способ питания: от сети/автономный.
3. Выводить оставшееся время до разрядки аккумулятора в случае использования автономного источника питания.
4. Выводить оставшееся время до зарядки аккумулятора, когда он подключён к сети.
5. Показывать статус зарядки батареи (заряжается/не заряжается).
6. Вывести максимальную вместительность батареи.
7. Вывести текущую вместительность батареи.
8. Выводить статус энергосберегающего режима.
9. Предоставлять возможность отправить компьютер в режим сна.
10. Предоставлять возможность отправить компьютер в режим гибернации.
11. Вывести тип аккумулятора.
12. Выводить циклы зарядки/разрядки, если батарея имеет счётчик данных циклов.

## 3.2 Программная реализация

Итоговый программный продукт представлен на рисунке 9.

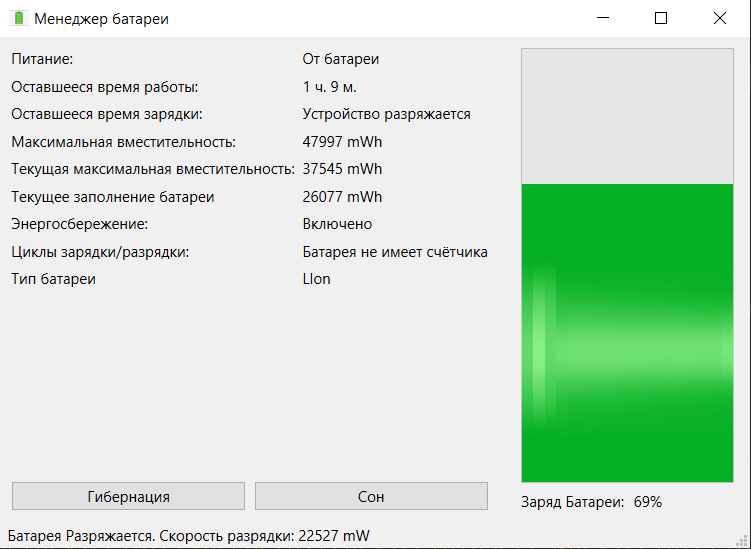


Рисунок 9 – интерфейс программного продукта.

Сама программа представлена одним окном класса MainWindow, которое в свою очередь наследуется от класса QMainWindow, и одним экземпляром написанного мной класса batteryManager.

Класс окна MainWindow реализован следующим образом:

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr){

bm = new batteryManager();

bm->fillInfrormation();

ui->setupUi(this);

fillingFields();

QTimer\* timer = new QTimer(this);

connect(timer, SIGNAL(timeout()), this, SLOT(onTimeout()));

timer->start(3000); // обновление каждые 3с, в идеале сделать обновление на изменение в структурах

}

~MainWindow(){

delete ui;

}

private slots:

void on\_btnSleep\_clicked(){

bm->sleep();

}

void on\_btnHibernation\_clicked(){

bm->hibernation();

}

public slots:

void onTimeout(){

bm->fillBattery();

bm->fillInfrormation();

bm->initChemistry();

fillingFields();

}

void fillingFields(){

ui->AC\_status\_value->setText(this->bm->getACStatus());

ui->battery\_charge\_level\_value->setText(this->bm->getBatteryChargeProcent() + "%");

ui->battery\_type\_value->setText(this->bm->getBatteryType());

ui->current\_capacity\_value->setText(this->bm->getCurrentCapacity() + " mWh");

ui->max\_capacity\_value->setText(this->bm->getMaxCapacity() + " mWh");

ui->designed\_capacity\_value->setText(this->bm->getDesignedCapacity() + " mWh");

ui->cycles\_value->setText(this->bm->getCyclesCount());

QString bat\_lifetime = this->bm->getBatteryLifetime();

if (bat\_lifetime == "От сети" && this->bm->getACStatus() == "От батареи"){

ui->remaning\_time\_work\_value->setText("...");

} else {

ui->remaning\_time\_work\_value->setText(bat\_lifetime);

}

ui->remaning\_time\_charge\_value->setText(this->bm->getBatteryRemaninChargeTime());

if (ui->remaning\_time\_work->text() == "От сети" &&

this->bm->getACStatus() == "От батареи"){

ui->remaning\_time\_work->setText("...");

}

ui->power\_saving\_value->setText(this->bm->getBatterySaverStatus());

if (this->bm->charging()){

statusBar()->showMessage(tr("Батарея Заряжается.") + " Скорость зарядки: " +

this->bm->getChargeSpeed().c\_str() + " mW");

} else if (this->bm->discharging()) {

statusBar()->showMessage(tr("Батарея Разряжается.") + " Скорость разрядки: " +

this->bm->getChargeSpeed().c\_str() + " mW");

} else {

statusBar()->showMessage(tr("Батарея Заряжена."));

}

int procent = this->bm->getBatteryProcent();

if(procent < 20){

ui->batteryBar->setValue(procent);

} else if (procent >= 20 && procent <=100) {

ui->batteryBar->setValue(procent);

} else {

// ui->batteryBar->setTextVisible(true); нужно вставить ошибку

}

private:

Ui::MainWindow \*ui;

batteryManager \*bm;

};

Таким образом, как видно из кода главного окна, вся логика работы с батареей инкапсулирована внутри класса batteryManager, откуда посредством get методов пользовательский интерфейс(UI) получает данные, для предоставления клиенту.

Класс batteryManager реализован следующим образом:

*class* batteryManager

{

*private*:

UCHAR chemistry[4] = { "0" };

SYSTEM\_POWER\_STATUS battery;

SYSTEM\_POWER\_STATE information;

BATTERY\_INFORMATION BatteryInfo;

ULONG DesignedCapacity;

ULONG CycleCount;

*public*:

batteryManager(){

bool **fill\_battery\_sucsess** = fillBattery();

fillInfrormation();

initChemistry();

}

void initChemistry(){

HDEVINFO **DeviceInfoSet**;

DeviceInfoSet = SetupDiGetClassDevs(&GUID\_DEVCLASS\_BATTERY, NULL, NULL, DIGCF\_PRESENT | DIGCF\_DEVICEINTERFACE);

SP\_DEVICE\_INTERFACE\_DATA **DeviceInterfaceData**;

DeviceInterfaceData.cbSize = *sizeof*(SP\_DEVINFO\_DATA);

SetupDiEnumDeviceInterfaces(*DeviceInfoSet*, NULL, &GUID\_DEVCLASS\_BATTERY, 0, &*DeviceInterfaceData*);

DWORD **cbRequired** = 0;

SetupDiGetDeviceInterfaceDetail(*DeviceInfoSet*, &*DeviceInterfaceData*, NULL, NULL, &*cbRequired*, NULL);

PSP\_DEVICE\_INTERFACE\_DETAIL\_DATA **devId** = (PSP\_DEVICE\_INTERFACE\_DETAIL\_DATA)LocalAlloc(LPTR, cbRequired);

devId->cbSize = *sizeof*(\*devId);

SetupDiGetDeviceInterfaceDetail(*DeviceInfoSet*, &*DeviceInterfaceData*, *devId*, cbRequired, &*cbRequired*, NULL);

HANDLE **hBattery** = CreateFile(devId->DevicePath, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

BATTERY\_QUERY\_INFORMATION **BatteryQueryInformation** = { 0 };

DWORD **bytesWait** = 0;

DWORD **bytesReturned** = 0;

DeviceIoControl(*hBattery*, IOCTL\_BATTERY\_QUERY\_TAG, &*bytesWait*, *sizeof*(*bytesWait*), &*BatteryQueryInformation*.*BatteryTag*,

*sizeof*(*BatteryQueryInformation*.*BatteryTag*), &*bytesReturned*, NULL) && BatteryQueryInformation.BatteryTag;

BATTERY\_INFORMATION **BatteryInfo** = { 0 };

BatteryQueryInformation.InformationLevel = *BatteryInformation*;

DeviceIoControl(*hBattery*, IOCTL\_BATTERY\_QUERY\_INFORMATION, &*BatteryQueryInformation*, *sizeof*(*BatteryQueryInformation*),

&*BatteryInfo*, *sizeof*(*BatteryInfo*), &*bytesReturned*, NULL);

*for* (int **b** = 0; b < 4; b++) {

chemistry[b] = BatteryInfo.Chemistry[b];

};

*this*->DesignedCapacity = BatteryInfo.DesignedCapacity;

*this*->CycleCount = BatteryInfo.CycleCount;

LocalFree(*devId*);

SetupDiDestroyDeviceInfoList(*DeviceInfoSet*);

}

void fillInfrormation(){

CallNtPowerInformation(*SystemBatteryState*, NULL, 0, &*this*->*information*, *sizeof*(*this*->*information*));

}

bool fillBattery(){

*return* GetSystemPowerStatus(&*this*->*battery*);

}

void sleep(){

SetSuspendState(*false*, *false*, *true*);

}

void hibernation(){

SetSuspendState(*true*, *false*, *false*);

}

QString getACStatus(){

*if* (*this*->battery.ACLineStatus == 1) {

*return* "От сети";

} *else* *if* (*this*->battery.ACLineStatus == 0){

*return* "От батареи";

} *else* {

*return* "Статус неизвестен";

}

}

QString getBatteryChargeProcent(){

int **battery\_life\_percent** = *this*->battery.BatteryLifePercent;

*return* QString::number(battery\_life\_percent);

}

QString getBatteryType(){

char **a**[4];

*for* (int **b** = 0; b < 4; b++)

{

a[b] = *this*->chemistry[b];

}

QString **battery\_type** = QString::fromUtf8(a);

*return* battery\_type;

}

QString getCurrentCapacity(){

QString **capacity** = QString::number(*this*->information.RemainingCapacity);

*return* capacity;

}

QString getMaxCapacity(){

QString **capacity** = QString::number(*this*->information.MaxCapacity);

*return* capacity;

}

QString getDesignedCapacity(){

QString **capacity** = QString::number(*this*->DesignedCapacity);

*return* capacity;

}

QString getCyclesCount(){

*if* (*this*->CycleCount == 0){

*return* "Батарея не имеет счётчика";

} *else* {

QString **cycles** = QString::number(*this*->CycleCount);

*return* cycles;

}

}

QString getBatteryLifetime(){

*if* (*this*->battery.BatteryLifeTime == -1){

*return* "От сети";

} *else* {

int **hours** = *this*->battery.BatteryLifeTime / 3600;

int **minuts** = (*this*->battery.BatteryLifeTime / 60) - 60 \* hours;

std::string **time** = std::to\_string(hours) + " ч. " + std::to\_string(minuts) + " м. ";

*return* QString::fromStdString(time);

}

}

QString getBatteryRemaninChargeTime(){

*if* (*this*->discharging()){

*return* "Устройство разряжается";

} *else* *if* (*this*->charging()){

int **div** = (*this*->getRate() / 60);

int **timezz** = (int)(*this*->getMaxCapacityInt() - *this*->getCurrentCapacityInt());

*if* (div != 0){

timezz /= div;

}

int **hours** = timezz / 60;

int **minuts** = timezz % 60;

std::string **time** = std::to\_string(hours) + " ч. " + std::to\_string(minuts) + " м. ";

*return* QString::fromStdString(time);

} *else* {

*return* "Не заряжается";

}

}

QString getBatterySaverStatus(){

*if* (*this*->battery.SystemStatusFlag == 0){

*return* "Выключено";

} *else* {

*return* "Включено";

}

}

std::string getChargeSpeed(){

*return* std::to\_string(abs(*this*->getRate()));

}

int getBatteryProcent(){

int **battery\_life\_percent** = *this*->battery.BatteryLifePercent;

*return* battery\_life\_percent;

}

int getMaxCapacityInt(){

int **capacity** = *this*->information.MaxCapacity;

*return* capacity;

}

int getCurrentCapacityInt(){

int **capacity** = *this*->information.RemainingCapacity;

*return* capacity;

}

long getRate(){

*return* *this*->information.Rate;

}

bool charging(){

*return* *this*->information.Charging;

}

bool discharging(){

*return* *this*->information.Discharging;

}

};

Камнем преткновения программной реализации стал счётчик циклов зарядки/разрядки, поскольку данный датчик установлен далеко не на всех батареях. Таким образом, даже сама операционная система Windows не всегда может предоставить пользователю количество циклов зарядки/разрядки.

Так же стоит отметить, что пункт «оставшееся время работы» является довольно условным показателем относительно реального времени оставшейся работы. Значительное расхождение данных цифр обусловлено постоянным изменением нагрузки системы на батарею. К примеру, различного рода компьютерные игры при использовании питания от сети значительным образом нагружают систему, отчего батареи хватит на непродолжительный промежуток времени.

# Архитектура разрабатываемой программы

Разработанная программа является QT-Desktop приложением. Структура приложения изображена на рисунке 10.

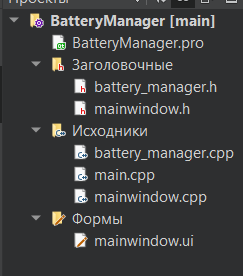


Рисунок 10 – структура проекта

BatteryManager – класс менеджера батареи, содержащий в себе всю логику взаимодействия с батареей. В объекте данного класса, который находится в объекте главного окна, содержится вся информация о встроенной в устройство батарее, если таковая имеется. Сам класс предоставляет «наружу» лишь get методы, благодаря которым графический интерфейс получает всю необходимую информацию о батарее для вывода на экран.

Mainwindow – класс главного окна, содержащий в себе слоты(обработчики), а так же объект менеджера батареи. Данный класс в методе заполнения полей(QLable) достаёт информацию о батарее из объекта менеджера батареи bm, инкапсулированного внутри, а затем заполняет все поля.

Функциональная схема проекта представлена на рисунке 11.



Рисунок 11 – Функциональная схема программы.

Функциональная схема отражает структуру приложения, которое состоит из двух подсистем: первая отвечает за получение данных от батареи, вторая – представляет собой графический интерфейс, который выводит все характеристики батареи на экран. Также подсистема графического интерфейса отвечает за обработку нажатия клавиш «сон», «гибернация», «закрыть».

Блок-схема программы представлена на рисунке 12.

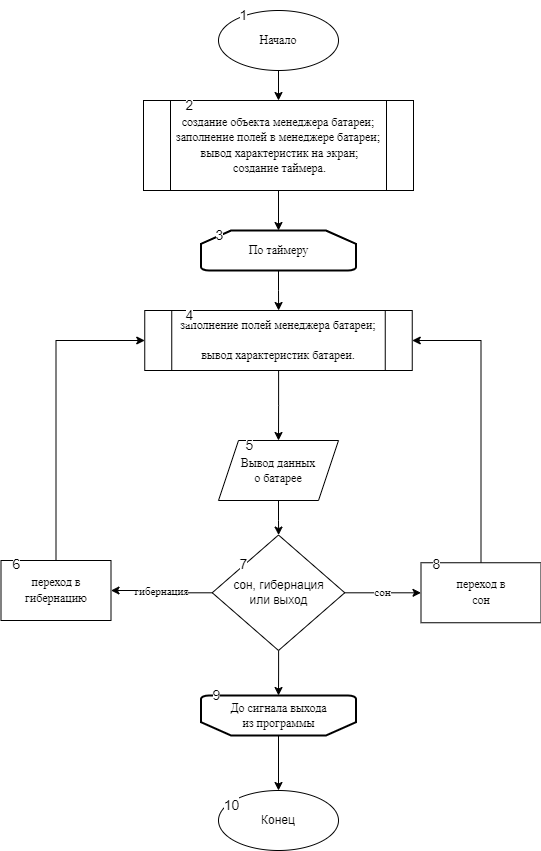


Рисунок 12 – Блок-схема алгоритма программы.

Алгоритм программы, согласно блок-схеме, следующий: программа запускается, после чего происходит создание объекта bm класса batteryManager, который будет хранить в себе все необходимые характеристики батареи. Затем поля объекта bm заполняются информацией, взятой из системных структур. После заполнения полей объекта bm, создаётся объект timer класса QTimer. Значение таймера устанавливается в 3000мс. Далее по каждому такту таймера происходит обновление полей объекта bm. Следом за обновлением полей происходит вывод данных о батарее пользователю. Далее у пользователя есть несколько опций, а конкретно – «сон», «гибернация», «закрыть». «Сон» отправляет компьютер в режим сна, «гибернация», соответственно, в режим гибернации, а «закрыть» - закрывает приложение. После пробуждения из режимов сна или гибернации происходит заполнение полей объекта bm, а дальше весь цикл повторяется заново.

# ЗАКЛЮЧЕНИЕ

Таким образом, в ходе выполнения данной курсовой работы был разработан программный продукт «Менеджер работы батареи ОС Windows».

В разработке данной программы были использованы набор программных средств Qt-Creator. Данные, касающиеся батареи были взяты из различных системных структур.

Был рассмотрен механизм взаимодействия с батареей устройства посредством использования вышеописанных структур, посредством языка C++.

И, безусловно, самым главным элементом данной курсовой работы является выходной программный продукт, предоставляющий пользователю удобный программный интерфейс, который даёт пользователю подробные данные о состоянии батареи.

Также присутствует возможности развития данного программного продукта в дальнейшем. К примеру, помимо добавления различного функционала, значительным улучшением данного проекта будет его правильное развёртывание: создание для программы инсталлятора и загрузки его на платформы распространения программных продуктов в интернете. Например, данный проект с открытым исходным кодом может быть размещён в моём github репозитории, откуда любой пользователь сможет скачать всего 1 файл – инсталлятор, а не целиком скомпилированную программу, в которой ещё нужно разобраться. После установки инсталлятора достаточно запустить его и выбрать директорию, куда будет установлен проект – и всё готово.

# СПИСОК ЛИТЕРАТУРЫ

[1] Microsoft [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winbase/ns-winbase-system\_power\_status.

[2] Microsoft [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/powerbase/nf-powerbase-callntpowerinformation?source=recommendations.

[3] Microsoft [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winnt/ne-winnt-system\_power\_state.

[4] Microsoft [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-system\_power\_capabilities.

[5] GitHub [Электронный ресурс]. – Режим доступа: https://github.com/MicrosoftDocs/sdk-api/blob/docs/sdk-api-src/content/winnt/ns-winnt-system\_battery\_state.md.

[6] GitHub [Электронный ресурс]. – Режим доступа: https://github.com/MicrosoftDocs/sdk-api/blob/docs/sdk-api-src/content/winnt/ns-winnt-system\_power\_level.md.

[7] Kharkov [Электронный ресурс]. – Режим доступа: https://vist.kharkov.ua/articles/komplektuyuschie-i-periferiya/kakie-bivayut-batarei-dlya-noutbuka.html. – Дата доступа: 07.12.22.

[8] Microsoft [Электронный ресурс]. – Режим доступа:

https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-power\_action\_policy.

[9] Microsoft [Электронный ресурс]. – Режим доступа:

https://learn.microsoft.com/en-us/windows/win32/api/batclass/ns-batclass-battery\_wmi\_cycle\_count.

[10] Microsoft [Электронный ресурс]. – Режим доступа:

https://learn.microsoft.com/en-us/previous-versions/ff536283(v=vs.85)

[11] Microsoft [Электронный ресурс]. – Режим доступа:

https://learn.microsoft.com/en-us/windows/win32/api/batclass/ns-batclass-battery\_wmi\_full\_charged\_capacity

[12] Внутреннее устройство Windows 7-е издание : справочник / Марк Руссинович [и др.]. – СПб.: Питер, 2018. – 944 с.: ил

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг кода**

Main.cpp:

#include "mainwindow.h"

#include <QApplication>

int **main**(int **argc**, char \***argv**[])

{

qputenv("QT\_AUTO\_SCREEN\_SCALE\_FACTOR","1");

QApplication **a**(*argc*, *argv*);

MainWindow **w**;

w.show();

*return* a.exec();

}

mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "battery\_manager.h"

#include "qtimer.h"

#include "QProgressBar"

QT\_BEGIN\_NAMESPACE

*namespace* Ui { *class* MainWindow; }

QT\_END\_NAMESPACE

*class* MainWindow : *public* QMainWindow

{

Q\_OBJECT

*public*:

MainWindow(QWidget \*parent = *nullptr*);

~MainWindow();

*private* *slots*:

void on\_btnSleep\_clicked();

void on\_btnHibernation\_clicked();

*public* *slots*:

void onTimeout();

void fillingFields();

*private*:

Ui::MainWindow \*ui;

batteryManager \*bm;

};

#endif *//* *MAINWINDOW\_H*

mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \***parent**)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

bm = *new* batteryManager();

bm->fillInfrormation();

ui->setupUi(*this*);

fillingFields();

QTimer\* **timer** = *new* QTimer(*this*);

connect(timer, SIGNAL(timeout()), *this*, SLOT(onTimeout()));

timer->start(3000); *//* *обновление* *каждые* *3с,* *в* *идеале* *сделать* *обновление* *на* *изменение* *в* *структурах*

}

MainWindow::~**MainWindow**()

{

*delete* ui;

}

void MainWindow::**on\_btnSleep\_clicked**()

{

bm->sleep();

}

void MainWindow::**on\_btnHibernation\_clicked**()

{

bm->hibernation();

}

void MainWindow::**onTimeout**()

{

bm->fillBattery();

bm->fillInfrormation();

bm->initChemistry();

fillingFields();

}

void MainWindow::**fillingFields**()

{

ui->AC\_status\_value->setText(*this*->bm->getACStatus());

ui->battery\_charge\_level\_value->setText(*this*->bm->getBatteryChargeProcent() + "%");

ui->battery\_type\_value->setText(*this*->bm->getBatteryType());

ui->current\_capacity\_value->setText(*this*->bm->getCurrentCapacity() + " mWh");

ui->max\_capacity\_value->setText(*this*->bm->getMaxCapacity() + " mWh");

ui->designed\_capacity\_value->setText(*this*->bm->getDesignedCapacity() + " mWh");

ui->cycles\_value->setText(*this*->bm->getCyclesCount());

QString **bat\_lifetime** = *this*->bm->getBatteryLifetime();

*if* (bat\_lifetime == "От сети" && *this*->bm->getACStatus() == "От батареи"){

ui->remaning\_time\_work\_value->setText("...");

} *else* {

ui->remaning\_time\_work\_value->setText(bat\_lifetime);

}

ui->remaning\_time\_charge\_value->setText(*this*->bm->getBatteryRemaninChargeTime());

*if* (ui->remaning\_time\_work->text() == "От сети" &&

*this*->bm->getACStatus() == "От батареи"){

ui->remaning\_time\_work->setText("...");

}

ui->power\_saving\_value->setText(*this*->bm->getBatterySaverStatus());

*if* (*this*->bm->charging()){

statusBar()->showMessage(tr("Батарея Заряжается.") + " Скорость зарядки: " +

*this*->bm->getChargeSpeed().c\_str() + " mW");

}*else* *if* (*this*->bm->discharging()) {

statusBar()->showMessage(tr("Батарея Разряжается.") + " Скорость разрядки: " +

*this*->bm->getChargeSpeed().c\_str() + " mW");

} *else* {

statusBar()->showMessage(tr("Батарея Заряжена."));

}

int **procent** = *this*->bm->getBatteryProcent();

*//int* *procent* *=* *10;*

*if*(procent < 20){

*//* *ui->batteryBar->setTextVisible(false);*

ui->batteryBar->setValue(procent);

*//* *ui->batteryBar->setStyleSheet("* *QProgressBar* *{* *border:* *2px* *solid* *grey;* *border-radius:* *0px;* *text-align:* *center;* *}* *QProgressBar::chunk* *{background-color:* *#FF0000;* *width:* *1px;}");*

} *else* *if* (procent >= 20 && procent <=100) {

ui->batteryBar->setValue(procent);

} *else* {

*//* *ui->batteryBar->setTextVisible(true);* *нужно* *вставить* *ошибку*

}

}

battery\_manager.h:

#ifndef BATTERY\_MANAGER\_H

#define BATTERY\_MANAGER\_H

#include <iostream>

#include <Windows.h> *//* *UCHAR*

#include <Powerbase.h>

#include <QString>

#include <winbase.h>

#pragma comment (lib, "setupapi.lib") *//* *for* *Chemistry*

#include <Poclass.h>

#include <Setupapi.h>

#include <devguid.h>

#pragma comment(lib, "PowrProf.lib")

*extern* "C" {

#include <Powrprof.h>

}

#include "batclass.h"

*class* batteryManager

{

*private*:

UCHAR chemistry[4] = { "0" };

SYSTEM\_POWER\_STATUS battery;

SYSTEM\_BATTERY\_STATE information;

BATTERY\_INFORMATION BatteryInfo;

ULONG DesignedCapacity;

ULONG CycleCount;

*public*:

batteryManager();

void initChemistry();

void fillInfrormation();

bool fillBattery();

void fillChemistry();

void sleep();

void hibernation();

QString getACStatus();

QString getBatteryChargeProcent();

QString getBatteryType();

QString getCurrentCapacity();

QString getMaxCapacity();

QString getDesignedCapacity();

QString getCyclesCount();

QString getBatteryLifetime();

QString getBatteryRemaninChargeTime();

QString getBatterySaverStatus();

std::string getChargeSpeed();

int getBatteryProcent();

int getMaxCapacityInt();

int getCurrentCapacityInt();

long getRate();

bool charging();

bool discharging();

};

#endif *//* *BATTERY\_MANAGER\_H*

battery\_manager.cpp:

#include "battery\_manager.h"

batteryManager::batteryManager()

{

bool fill\_battery\_sucsess = fillBattery();

fillInfrormation();

initChemistry();

}

void batteryManager::initChemistry(){

HDEVINFO DeviceInfoSet;

DeviceInfoSet = SetupDiGetClassDevs(&GUID\_DEVCLASS\_BATTERY, NULL, NULL, DIGCF\_PRESENT | DIGCF\_DEVICEINTERFACE);

SP\_DEVICE\_INTERFACE\_DATA DeviceInterfaceData;

DeviceInterfaceData.cbSize = *sizeof*(SP\_DEVINFO\_DATA);

SetupDiEnumDeviceInterfaces(DeviceInfoSet, NULL, &GUID\_DEVCLASS\_BATTERY, 0, &DeviceInterfaceData);

DWORD cbRequired = 0;

SetupDiGetDeviceInterfaceDetail(DeviceInfoSet, &DeviceInterfaceData, NULL, NULL, &cbRequired, NULL);

PSP\_DEVICE\_INTERFACE\_DETAIL\_DATA devId = (PSP\_DEVICE\_INTERFACE\_DETAIL\_DATA)LocalAlloc(LPTR, cbRequired);

devId->cbSize = *sizeof*(\*devId);

SetupDiGetDeviceInterfaceDetail(DeviceInfoSet, &DeviceInterfaceData, devId, cbRequired, &cbRequired, NULL);

HANDLE hBattery = CreateFile(devId->DevicePath, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

BATTERY\_QUERY\_INFORMATION BatteryQueryInformation = { 0 };

DWORD bytesWait = 0;

DWORD bytesReturned = 0;

DeviceIoControl(hBattery, IOCTL\_BATTERY\_QUERY\_TAG, &bytesWait, *sizeof*(bytesWait), &BatteryQueryInformation.BatteryTag,

*sizeof*(BatteryQueryInformation.BatteryTag), &bytesReturned, NULL) && BatteryQueryInformation.BatteryTag;

BATTERY\_INFORMATION BatteryInfo = { 0 };

BatteryQueryInformation.InformationLevel = BatteryInformation;

DeviceIoControl(hBattery, IOCTL\_BATTERY\_QUERY\_INFORMATION, &BatteryQueryInformation, *sizeof*(BatteryQueryInformation),

&BatteryInfo, *sizeof*(BatteryInfo), &bytesReturned, NULL);

*for* (int b = 0; b < 4; b++) {

chemistry[b] = BatteryInfo.Chemistry[b];

};

*this*->DesignedCapacity = BatteryInfo.DesignedCapacity;

*this*->CycleCount = BatteryInfo.CycleCount;

LocalFree(devId);

SetupDiDestroyDeviceInfoList(DeviceInfoSet);

}

void batteryManager::fillInfrormation(){

CallNtPowerInformation(SystemBatteryState, NULL, 0, &*this*->information, *sizeof*(*this*->information));

}

bool batteryManager::fillBattery()

{

*return* GetSystemPowerStatus(&*this*->battery);

}

void batteryManager::sleep(){

SetSuspendState(*false*, *false*, *true*);

}

void batteryManager::hibernation(){

SetSuspendState(*true*, *false*, *false*);

}

QString batteryManager::getACStatus()

{

*if* (*this*->battery.ACLineStatus == 1) {

*return* "От сети";

} *else* *if* (*this*->battery.ACLineStatus == 0){

*return* "От батареи";

} *else* {

*return* "Статус неизвестен";

}

}

QString batteryManager::getBatteryChargeProcent(){

int battery\_life\_percent = *this*->battery.BatteryLifePercent;

*return* QString::number(battery\_life\_percent);

}

QString batteryManager::getBatteryType()

{

char a[4];

*for* (int b = 0; b < 4; b++)

{

a[b] = *this*->chemistry[b];

}

QString battery\_type = QString::fromUtf8(a);

*return* battery\_type;

}

QString batteryManager::getCurrentCapacity()

{

QString capacity = QString::number(*this*->information.RemainingCapacity);

*return* capacity;

}

QString batteryManager::getMaxCapacity()

{

QString capacity = QString::number(*this*->information.MaxCapacity);

*return* capacity;

}

QString batteryManager::getDesignedCapacity()

{

QString capacity = QString::number(*this*->DesignedCapacity);

*return* capacity;

}

QString batteryManager::getCyclesCount(){

*if* (*this*->CycleCount == 0){

*return* "Батарея не имеет счётчика";

} *else* {

QString cycles = QString::number(*this*->CycleCount);

*return* cycles;

}

}

QString batteryManager::getBatteryLifetime()

{

*if* (*this*->battery.BatteryLifeTime == -1){

*return* "От сети";

} *else* {

int hours = *this*->battery.BatteryLifeTime / 3600;

int minuts = (*this*->battery.BatteryLifeTime / 60) - 60 \* hours;

std::string time = std::to\_string(hours) + " ч. " + std::to\_string(minuts) + " м. ";

*return* QString::fromStdString(time);

}

}

QString batteryManager::getBatteryRemaninChargeTime()

{

*if* (*this*->discharging()){

*return* "Устройство разряжается";

} *else* *if* (*this*->charging()){

int div = (*this*->getRate() / 60);

int timezz = (int)(*this*->getMaxCapacityInt() - *this*->getCurrentCapacityInt());

*if* (div != 0){

timezz /= div;

}

int hours = timezz / 60;

int minuts = timezz % 60;

std::string time = std::to\_string(hours) + " ч. " + std::to\_string(minuts) + " м. ";

*return* QString::fromStdString(time);

} *else* {

*return* "Не заряжается";

}

}

QString batteryManager::getBatterySaverStatus()

{

*if* (*this*->battery.SystemStatusFlag == 0){

*return* "Выключено";

} *else* {

*return* "Включено";

}

}

std::string batteryManager::getChargeSpeed()

{

*return* std::to\_string(abs(*this*->getRate()));

*//return* *std::to\_string(this->information.Rate);* *//* *неверно,* *так* *как* *DWORD*

}

int batteryManager::getBatteryProcent()

{

int battery\_life\_percent = *this*->battery.BatteryLifePercent;

*return* battery\_life\_percent;

}

int batteryManager::getMaxCapacityInt()

{

int capacity = *this*->information.MaxCapacity;

*return* capacity;

}

int batteryManager::getCurrentCapacityInt()

{

int capacity = *this*->information.RemainingCapacity;

*return* capacity;

}

long batteryManager::getRate()

{

*//long* *long* *a* *=* *this->information.Rate;*

*return* *this*->information.Rate;

}

bool batteryManager::charging()

{

*return* *this*->information.Charging;

}

bool batteryManager::discharging()

{

*return* *this*->information.Discharging;

}

# ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Графический интерфейс**

# ПРИЛОЖЕНИЕ В

**(обязательное)**

**Схема алгоритма**

# ПРИЛОЖЕНИЕ Г

**(обязательное)**

**Функциональная схема**