

СОДЕРЖАНИЕ

Введение.....	4
1 Платформа программного обеспечения.....	5
1.1 Историческая справка о платформе разработки Qt.....	5
2 Теоретическое обоснование разработки программного продукта.....	9
2.1 Оценка и анализ аналогов разрабатываемому программному продукту.....	9
2.2 Аппаратные характеристики персональных компьютеров	13
3 Проектирование функциональных возможностей программы	23
3.1 Обзор наиболее используемых подходов в проектировании программных продуктов.....	23
3.2 Структурное проектирование программного продукта.....	23
3.3 Объектно-ориентированное проектирование программного продукта	28
4 Архитектура разрабатываемой программы.....	33
4.1 Файловая структура разрабатываемого программного продукта.....	33
4.2 Диаграмма классов разрабатываемого программного продукта	36
4.3 Описание блок-схемы алгоритма приложения	37
Заключение	38
Список литературных источников	39
Приложение А (обязательное) Листинг кода программы.....	41
Приложение Б (обязательное) Блок-схема алгоритма приложения	59
Приложение В (обязательное) Функциональная схема приложения	60
Приложение Г (обязательное) Графический интерфейс приложения.....	61
Приложение Д (обязательное) Ведомость.....	62

ВВЕДЕНИЕ

Таск-менеджер (диспетчер задач, системный монитор) – менеджер запуска процессов и потоков, который поставляется вместе с используемой ОС, а также может быть установлен из сторонних источников сети Интернет. Он предоставляет информацию о производительности компьютера и запущенных приложениях, процессах и использовании ЦП, фиксирует нагрузку и сведения о памяти, сетевой активности и статистике, зарегистрированных пользователях и системных службах. Диспетчер задач также может использоваться для установки приоритетов процессов, свойства процессора, запуска и остановки служб и принудительного завершения процессов.

Цели курсового проекта:

1 Приобретение теоретических и практических навыков в разработке программного обеспечения на языке программирования *C++* и платформе *Qt*.

2 Ознакомление с разновидностями аппаратных характеристик персональных компьютеров, а также приобретение практических навыков оценки сложности получения тех или иных характеристик.

Задачи курсового проекта:

1 Оценка и анализ аналогичных программных продуктов, выделение их достоинств и недостатков.

2 Разработка архитектуры программного продукта.

3 Разработка пользовательского интерфейса.

4 Разработка основных функциональных элементов программного обеспечения.

1 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Для разработки выбранного программного обеспечения была выбрана платформа для разработки компьютерных приложений Qt, которая может использовать языки программирования Python/C++.

1.1 ИСТОРИЧЕСКАЯ СПРАВКА О ПЛАТФОРМЕ РАЗРАБОТКИ QT

Qt — фреймворк для разработки кроссплатформенного программного обеспечения на языке программирования C++. Впервые был выпущен в массовое использование 20 мая 1995 компанией Trolltech. Сам фреймворк также был написан на языке программирования C++. Компании, разработчики данной платформы: Trolltech (1991–2008), Nokia (2008–2011), Qt Project (2011 - ?), Digia (2012–2014), The Qt Company (2014 - настоящее время).

Существуют версии библиотеки для Microsoft Windows, систем класса UNIX с графической подсистемой X11, Android, iOS, Mac OS X, Microsoft Windows CE, QNX, встраиваемых Linux-систем и платформы S60. Идет портирование на Windows Phone и Windows RT. Также идёт портирование на Haiku и Tizen.

Некоторое время библиотека также распространялась ещё в версии Qt/Embedded, предназначенной для применения на встраиваемых и мобильных устройствах, но начиная с середины 2000-х годов она выделена в самостоятельный продукт Qtopia.

Начиная с версии 4.5 Qt распространяется по трём лицензиям:

Qt Commercial — для разработки программного обеспечения с собственной лицензией, допускающая модификацию самой Qt без раскрытия изменений;

GNU GPL — для разработки с открытыми исходниками, распространяемыми на условиях GNU GPL, а также для модификации Qt;

GNU LGPL — для разработки программного обеспечения с собственной лицензией.

Функции и состав Qt. Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-

ориентированным, расширяемым и поддерживающим технику компонентного программирования.

Отличительная особенность — использование метаобъектного компилятора — предварительной системы обработки исходного кода. Расширение возможностей обеспечивается системой плагинов, которые возможно размещать непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Комплектуется визуальной средой разработки графического интерфейса Qt Designer, позволяющей создавать диалоги и формы в режиме WYSIWYG. В поставке Qt есть Qt Linguist — графическая утилита, позволяющая упростить локализацию и перевод программы на многие языки; и Qt Assistant — справочная система Qt, упрощающая работу с документацией по библиотеке, а также позволяющая создавать кроссплатформенную справку для разрабатываемого на основе Qt программного обеспечения. Начиная с версии 4.5.0 в комплект включена среда разработки Qt Creator, которая включает редактор кода, справку, графические средства Qt Designer и возможность отладки приложений. Qt Creator может использовать GCC или Microsoft VC++ в качестве компилятора и GDB в качестве отладчика. Для Windows-версий библиотека комплектуется компилятором, заголовочными и объектными файлами MinGW.

Библиотека разделена на несколько модулей:

- QtCore — классы ядра библиотеки, используемые другими модулями;
- QtGui — компоненты графического интерфейса;
- QtNetwork — набор классов для сетевого программирования;
- QtOpenGL — набор классов для работы с OpenGL.
- QSql — набор классов для работы с базами данных с использованием языка структурированных запросов SQL;
- QtScript — классы для работы с Qt Scripts;
- QtSvg — классы для отображения и работы с данными Scalable Vector Graphics (SVG);
- QtXml — модуль для работы с XML, поддерживается SAX и DOM модели работы;
- QtDesigner — классы создания расширений QtDesigner для своих собственных виджетов;
- QtUiTools — классы для обработки в приложении форм Qt Designer;
- QtAssistant — справочная система;
- Qt3Support — модуль с классами, необходимыми для совместимости с библиотекой Qt версии 3.x.x;

- QtTest — модуль для работы с UNIT тестами;
- QtWebKit — модуль WebKit, интегрированный в Qt и доступный через её классы;
- QtXmlPatterns — модуль для поддержки XQuery 1.0 и XPath 2.0;
- Phonon — модуль для поддержки воспроизведения и записи видео и аудио, как локально, так и с устройств и по сети;
- QtCLucene — модуль для поддержки полнотекстового поиска, применяется в новой версии Assistant в Qt 4.4;
- ActiveQt — модуль для работы с ActiveX и COM технологиями для Qt разработчиков под Windows;
- QtDeclarative — модуль, предоставляющий декларативный фреймворк для создания динамических, настраиваемых пользовательских интерфейсов.

В Qt используется CamelCasing: имена классов выглядят как `MyClassName`, а имена методов – как `myMethodName`. При этом имена всех классов Qt начинаются с Q, например `QObject`, `QList` или `QFont`. Большинству классов соответствуют заголовочные файлы с тем же именем (без расширения `.h`).

Для эффективной работы с классами на стадии выполнения в Qt используется специальная объектная модель, расширяющая модель C++. В частности, добавляются следующие возможности: – древовидные иерархии объектов; – аналог `dynamic_cast` для библиотеки, не использующий RTTI; – взаимодействие объектов через сигналы и слоты; – свойства объектов. Многие объекты определяются значением сразу нескольких свойств, внутренними состояниями и связями с другими объектами. Они представляют собой индивидуальные сущности, и для них не имеет смысла операция буквального копирования, а также разделение данных в памяти. В Qt эти объекты наследуют свойства `QObject`. В тех случаях, когда объект требовалось бы рассматривать не как сущность, а как значение (например, при хранении в контейнере) – используются указатели. Иногда указатель на объект, наследуемый от `QObject`, называют просто объектом.

При создании графических пользовательских интерфейсов взаимодействие объектов часто осуществляется через обратные вызовы, т.е. передачу кода для последующего выполнения (в виде указателей на функции, функторов, и т.д.). Также популярна концепция событий и обработчиков, в которой обработчик действует как перехватчик события определенного объекта. В Qt вводится концепция сигналов и слотов. Сигнал отправляется при вызове соответствующего ему метода. Программисту при этом нужно только указать прототип метода в разделе `signals`.

Слот является методом, исполняемым при получении сигнала. Слоты могут объявляться в разделе `public slots`, `protected slots` или `private slots`. При этом

уровень защиты влияет лишь на возможность вызова слотов в качестве обычных методов, но не на возможность подключения сигналов к слотам. Модель сигналов и слотов отличается от модели событий и обработчиков тем, что слот может подключаться к любому числу сигналов, а сигнал может подключаться к любому числу слотов. При отправке сигнала будут вызваны все подключенные к нему слоты. Таким образом, для эффективной работы с классами на стадии выполнения Qt использует специальную объектную модель, в которой при помощи наследования от `QObject` и генерирования кода компилятором метаобъектов реализованы: – иерархии объектов; – специальный аналог `dynamic_cast`, не зависящий от RTTI; – система сигналов и слотов; – система свойств объектов; – динамическая работа с классами.

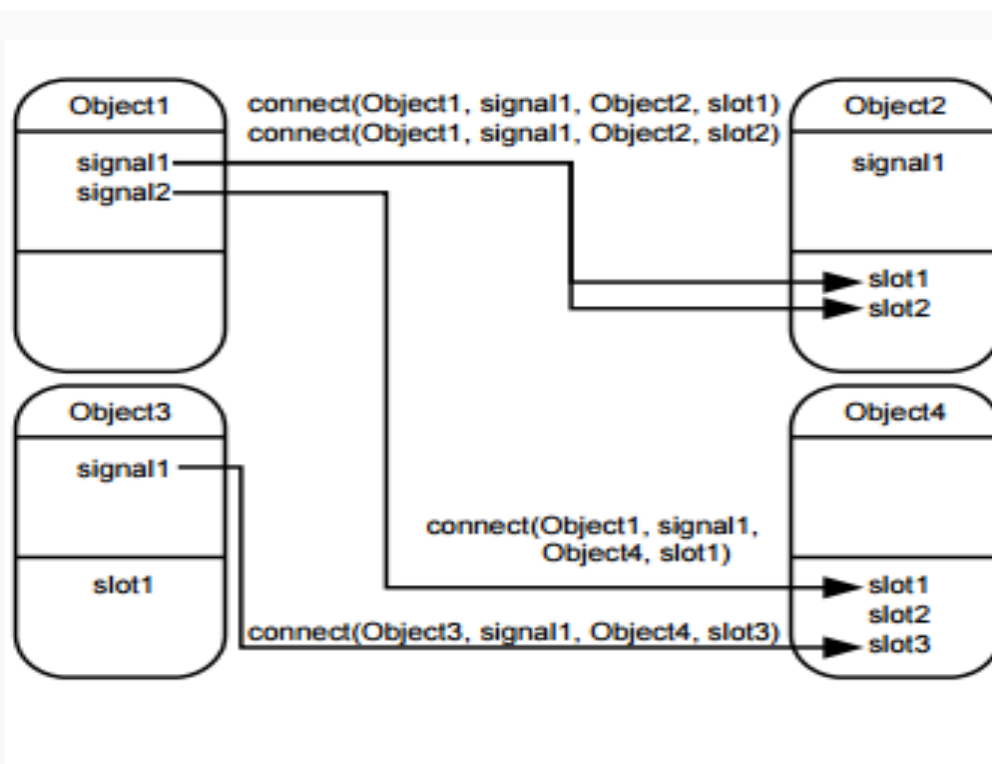


Рисунок 1.1 – схема связывания сигналов и слотов объекта

2 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

Среди аналогов разрабатываемого программного продукта наблюдается недостаток тех или иных функциональных возможностей. При необходимости получения информации об аппаратных характеристиках персонального компьютера, приходится использовать отдельные программные продукты, которые могут отражать конкретные показатели, но при необходимости работы с процессами ОС Windows мы должны пользоваться другим программным обеспечением, будь то продукты, поставляемые вместе с ОС или сторонние продукты. Исходя из перечисленных требований наблюдается потребность в программном продукте, который может одновременно отображать аппаратные характеристики персонального компьютера, а также быть монитором системных ресурсов.

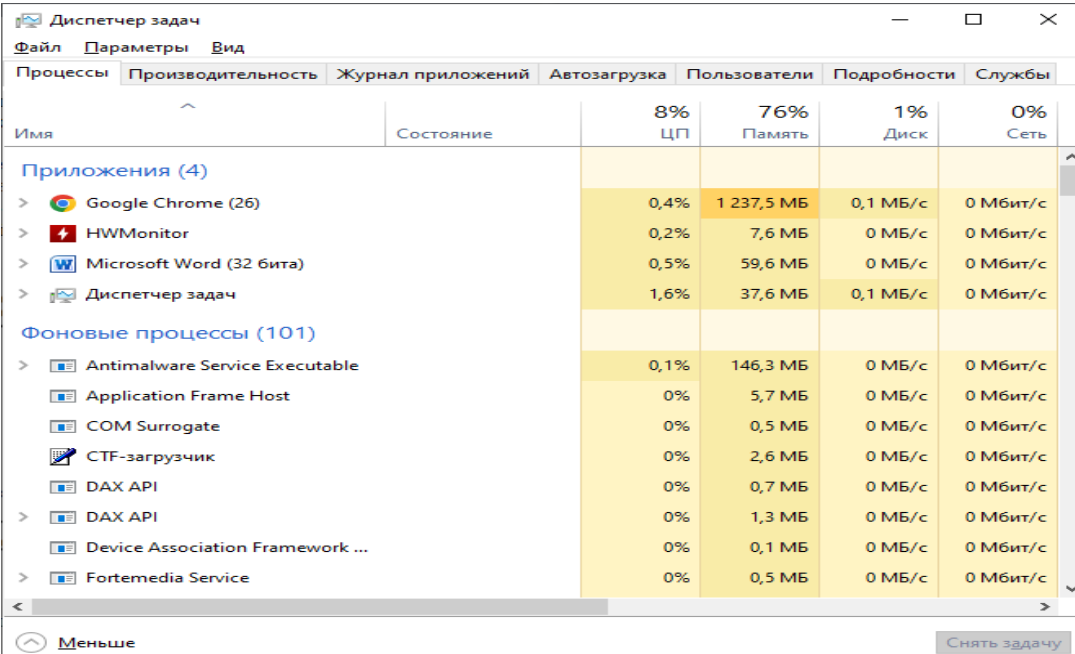
2.1 ОЦЕНКА И АНАЛИЗ АНАЛОГОВ РАЗРАБАТЫВАЕМОМУ ПРОГРАММНОМУ ПРОДУКТУ

В данном разделе рассмотрим программное обеспечение, которое обладает схожим функционалом с разрабатываемым продуктом, рассмотрим и проанализируем его функционал, а также выявим достоинства и недостатки рассматриваемых продуктов.

2.1.1 ДИСПЕТЧЕР ЗАДАЧ WINDOWS

Диспетчер задач Windows — диспетчер задач, системный монитор и менеджер запуска, входящий в состав ОС Windows. Он предоставляет информацию о производительности компьютера и запущенных приложениях, процессах и использовании ЦП, фиксирует нагрузку и сведения о памяти, сетевой активности и статистике, зарегистрированных пользователях и системных службах. Диспетчер задач также может использоваться для установки приоритетов процессов, свойства процессора, запуска и остановки служб и принудительного завершения процессов. Диспетчер задач был представлен в его текущем виде начиная с Windows NT 4.0. Предыдущие версии Windows NT, а также Windows 3.x, включают в себя приложение «Список задач», способное перечислять текущие процессы и завершать их или создавать новый процесс. Windows 9x имеет программу, известную как «Завершение работы программы» («Close Program»), в которой перечислены текущие запущенные программы, а также варианты закрытия программ и

выключение компьютера. Графический интерфейс данного приложения представлен на рисунке 2.1.



The screenshot shows the Windows Task Manager window with the 'Производительность' (Performance) tab selected. The window title is 'Диспетчер задач' (Task Manager). The menu bar includes 'Файл', 'Параметры', and 'Вид'. The tabs at the top are 'Процессы', 'Производительность' (active), 'Журнал приложений', 'Автозагрузка', 'Пользователи', 'Подробности', and 'Службы'. The main area displays system resource usage: CPU (8%), Memory (76%), Disk (1%), and Network (0%). Below this, a list of processes is shown, categorized into 'Приложения (4)' (Applications) and 'Фоновые процессы (101)' (Background processes). The list includes icons, names, and resource usage for each process.

Имя	Состояние	8% ЦП	76% Память	1% Диск	0% Сеть
Приложения (4)					
Google Chrome (26)		0,4%	1 237,5 МБ	0,1 МБ/с	0 Мбит/с
HWMonitor		0,2%	7,6 МБ	0 МБ/с	0 Мбит/с
Microsoft Word (32 бита)		0,5%	59,6 МБ	0 МБ/с	0 Мбит/с
Диспетчер задач		1,6%	37,6 МБ	0,1 МБ/с	0 Мбит/с
Фоновые процессы (101)					
Antimalware Service Executable		0,1%	146,3 МБ	0 МБ/с	0 Мбит/с
Application Frame Host		0%	5,7 МБ	0 МБ/с	0 Мбит/с
COM Surrogate		0%	0,5 МБ	0 МБ/с	0 Мбит/с
CTF-загрузчик		0%	2,6 МБ	0 МБ/с	0 Мбит/с
DAX API		0%	0,7 МБ	0 МБ/с	0 Мбит/с
DAX API		0%	1,3 МБ	0 МБ/с	0 Мбит/с
Device Association Framework ...		0%	0,1 МБ	0 МБ/с	0 Мбит/с
Fortemedia Service		0%	0,5 МБ	0 МБ/с	0 Мбит/с

At the bottom left, there is a 'Меньше' (Show less) button. At the bottom right, there is a 'Снять задачу' (End task) button.

Рисунок 2.1 – графический интерфейс Диспетчера задач Windows

Программный продукт нам предлагает следующие вкладки, раскрывающие основные функциональные особенности программы:

- процессы;
- производительность;
- журнал приложений
- автозагрузка;
- пользователи;
- подробности;
- службы.

Процессы. Во вкладке «Процессы» отображается список всех запущенных процессов в системе. Список включает службы Windows и процессы из других учетных записей. До Windows XP имена процессов длиной более 15 символов укорачивались. Начиная с Windows XP, клавиша Delete также может использоваться для завершения процессов во вкладке «Процессы». Щелчок правой кнопкой мыши по процессу в списке позволяет изменить приоритет процесса, установив свойства процессора (установив, какие ядра (или потоки) процессора могут выполнять процесс), и позволяет завершить процесс. Выбор опции «Завершить процесс» позволяет немедленно «убить» процесс. Выбор «Завершить дерево процессов» заставляет Windows немедленно

«убивать» процесс, а также все процессы, прямо или косвенно начатые этим процессом. В отличие от выбора «Завершить задачу» во вкладке «Приложения» при выборе «Завершить процесс» программе не сообщается предупреждение и нет возможности завершить ее до принудительного завершения процесса. Однако, когда процесс, который выполняется в контексте безопасности отличается от процесса, который выполнил вызов `TerminateProcess`, использование служебной команды `KILL` в командной строке является обязательным. По умолчанию во вкладке «процессы» отображается учетная запись пользователя, от имени которой выполняется процесс, нагрузка на процессор и оперативную память. Есть много других столбцов, которые можно настроить, выбрав «Выбрать столбцы» щелкнув правой клавишей мыши под одному из столбцов.

Производительность. Во вкладке «Производительность» отображаются общие статистические данные о производительности системы, в частности общее количество использования ЦП и объем памяти. Показана диаграмма недавнего использования для обоих этих значений. Также показаны дополнительная информация об оперативной памяти. Общее представление графического интерфейса представлено на рисунке 2.2.

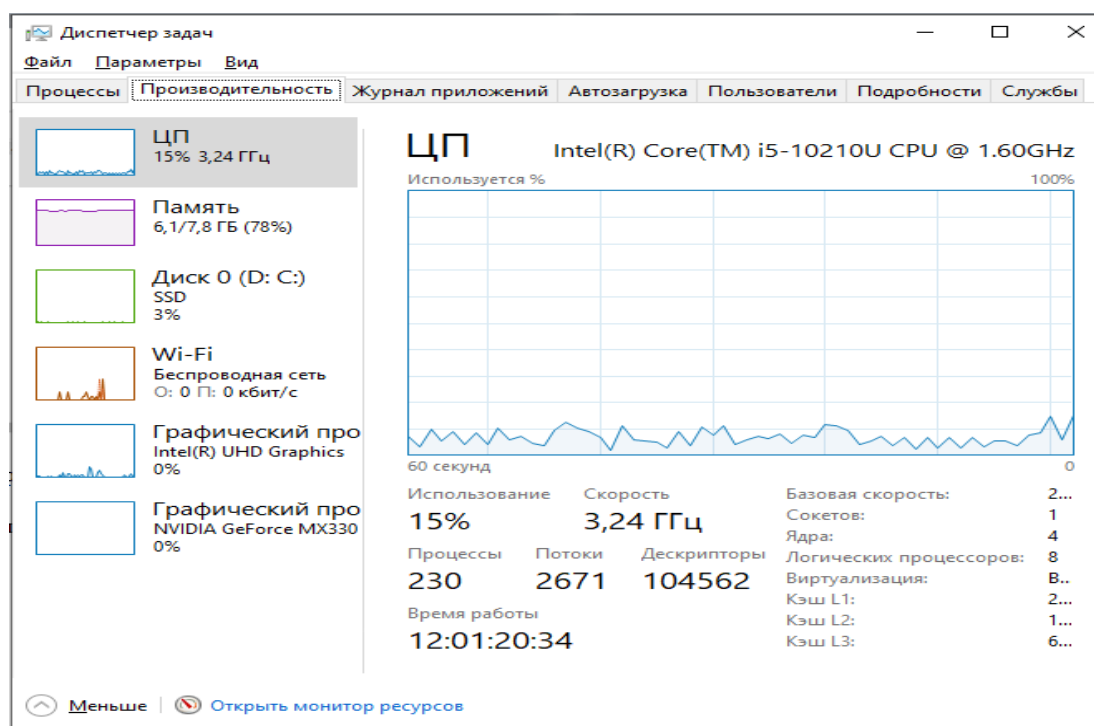


Рисунок 2.2 – графический интерфейс вкладки Производительность

Существует возможность разделить график использования ЦП на два: время режима ядра и время пользовательского режима. Многие драйверы устройств и основные компоненты операционной системы работают в режиме ядра, тогда как пользовательские приложения запускаются в пользовательском режиме. Параметр можно включить, кликнув правой клавишей мыши по графику и выбрав пункт «Показать время ядра». Когда эта опция включена, на графике использования ЦП будет показана бледно-голубая и темно-голубая область. Тёмно-голубая область — это количество времени, затраченного в режиме ядра, а бледно-голубая область показывает количество времени, проведенного в пользовательском режиме.

Сеть. Во вкладке «Журнал Приложений», представленной в Windows XP, отображается статистика, относящаяся к каждому сетевому адаптеру, присутствующему на компьютере. По умолчанию отображаются имя адаптера, процент использования сети, скорость соединения и состояние сетевого адаптера, а также диаграмма последней активности. Дополнительные параметры можно отобразить, выбрав «Выбрать столбцы» в меню «Вид».

Пользователи. Вкладка «Пользователи», также представленная в Windows XP, показывает всех пользователей, которые в настоящее время имеют сеанс на компьютере. На серверах может быть несколько пользователей, подключенных к компьютеру с помощью служб терминалов. В Windows XP может быть подключено одновременно несколько пользователей с помощью функции быстрого переключения пользователей как представлено на рисунке 2.3. Пользователи могут быть отключены или выведены из этой вкладки.

Исходя из функциональных возможностей данного программного обеспечения мы можем сделать вывод, что Диспетчер задач Windows отлично справляется с работой по управлению процессами и потоками, но имеет некоторые недостатки в информировании пользователя об аппаратных характеристиках персонального компьютера.

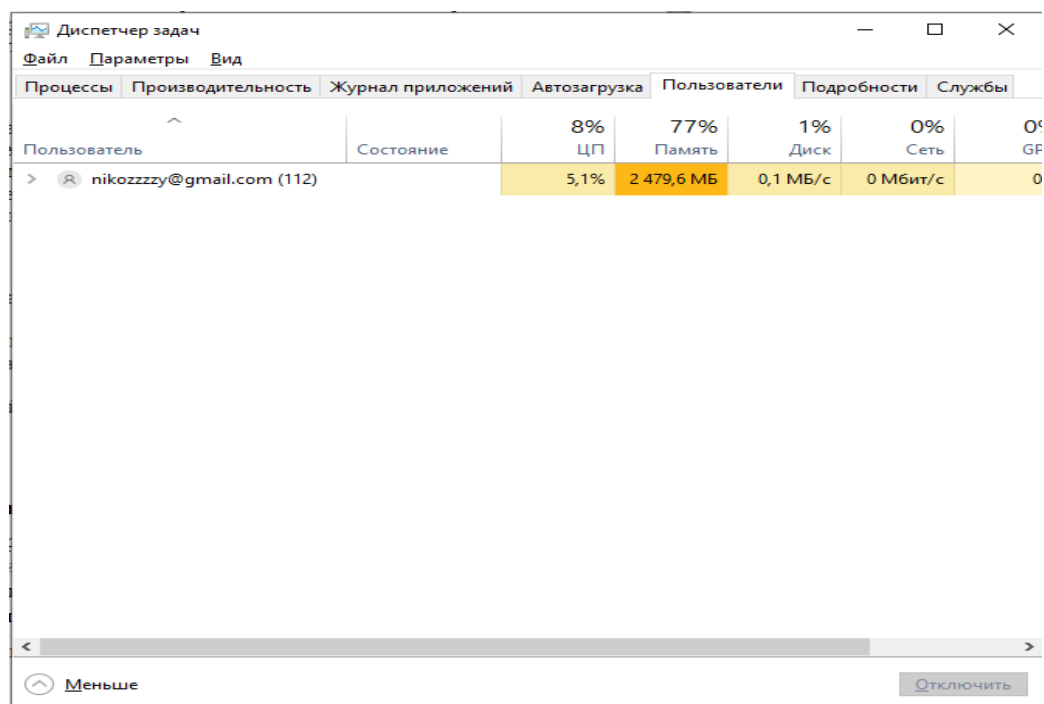


Рисунок 2.3 – графический интерфейс вкладки Пользователи

2.2 АППАРАТНЫЕ ХАРАКТЕРИСТИКИ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Персональный компьютер – это настольная или переносная ЭВМ, удовлетворяющая требованиям общедоступности и универсальности применения. Его аппаратную конфигурацию (состав оборудования) можно гибко изменять в соответствии с требованиями пользователя.

В состав персонального компьютера обязательно входят следующие устройства:

- системный блок;
- монитор;
- клавиатура;
- мышь.

2.2.1 СИСТЕМНЫЙ БЛОК

Системный блок – функциональный элемент, защищающий внутренние компоненты от внешнего воздействия и механических повреждений, поддерживающий необходимый температурный режим внутри, экранирующий

создаваемое внутренними компонентами электромагнитное излучение. Является основой для дальнейшего расширения системы.

В состав системного блока обязательно входят следующие устройства:

- материнская плата;
- процессор;
- оперативная память;
- жесткий диск;
- видеокарта;
- блок питания.

2.2.2 МАТЕРИНСКАЯ ПЛАТА

Материнская плата (motherboard), или системная плата (system board) – центральная комплексная плата, предоставляющая электронную и логическую связь между всеми устройствами, входящими в состав персонального компьютера. Пример материнской платы представлен на рисунке 2.4.



Рисунок 2.4 – материнская плата

На материнской плате располагаются основные электронные элементы компьютера:

Микропроцессор (CPU — Central Processing Unit — центральное вычислительное устройство, центральный процессор) устанавливается в специальный разъем типа ZIF1 (сокет), позволяющий заменить процессор без

специального инструмента. Каким образом чипсет представлен в материнской плате можно увидеть на рисунке 2.5.

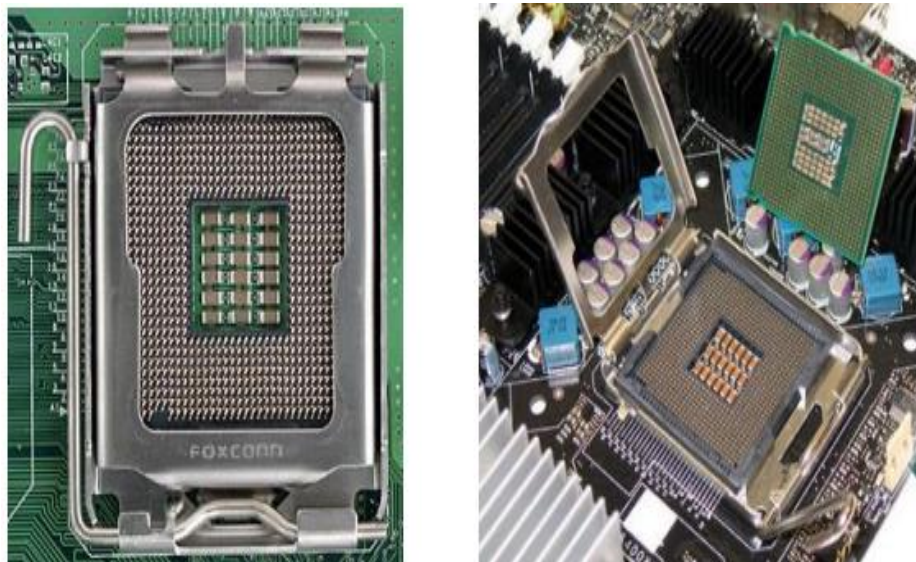


Рисунок 2.5 – socket для установки микропроцессора

1 Чипсет (chipset) — набор чипов (микросхем), управляющий взаимодействием процессора с другими устройствами. Чипсет полностью определяет все потенциальные возможности материнской платы: используемый процессор, тип и объем памяти, допустимые периферийные устройства.

2 Системная шина (system bus) – электрические соединения, по которым устройства компьютера обмениваются сигналами друг с другом. Все внешние устройства подключаются к шине непосредственно через соответствующие унифицированные разъемы (слоты) или через специфические адаптеры (контроллеры). Скорость (пропускная способность) системной шины влияет на скорость работы компьютера.

Микросхема постоянной памяти (ROM – Read Only Memory – память только для чтения), содержащая набор основных параметров компьютера, необходимых для совместной работы всех входящих в него устройств, и базовую систему ввода-вывода (Basic Input Output System – BIOS). Содержимое постоянной памяти поддерживается питанием от специальной батарейки.

Оперативная память (RAM — Random Access Memory — память с произвольным доступом) реализуется в виде модулей с микросхемами динамической памяти, которые вставляются в специальные разъемы на материнской плате (слоты) Как в материнской плате расположены слоты, представлено на рисунке 2.6.



Рисунок 2.6 – слоты для установки оперативной памяти

Кэш-память (cache) – очень быстрая (сверхоперативная) память, которая содержит информацию, необходимую процессору в первую очередь.

2.2.3 МИКРОПРОЦЕССОР

Микропроцессор — небольшая электронная схема в пластиковом или металлическом корпусе (размер менее 20 см²), которая выполняет все вычисления, пересылает данные между внутренними регистрами и управляет ходом вычислительного процесса. Именно процессор отвечает за обработку всех данных в системе и глобально управляет работой аппаратных устройств.

Конструктивно процессор состоит из ячеек, похожих на ячейки оперативной памяти. Внутренние ячейки процессора называют регистрами. В регистрах размещаются и данные и команды. С остальными устройствами компьютера, и в первую очередь оперативной памятью, процессор связан несколькими группами проводников, называемых шинами. Как выглядит микропроцессор представлено на рисунке 2.7.



Рисунок 2.7 – микропроцессор Intel

Основных шин три: **шина данных, адресная шина и командная шина**. Их связь с микропроцессором отражена на рисунке 2.8.

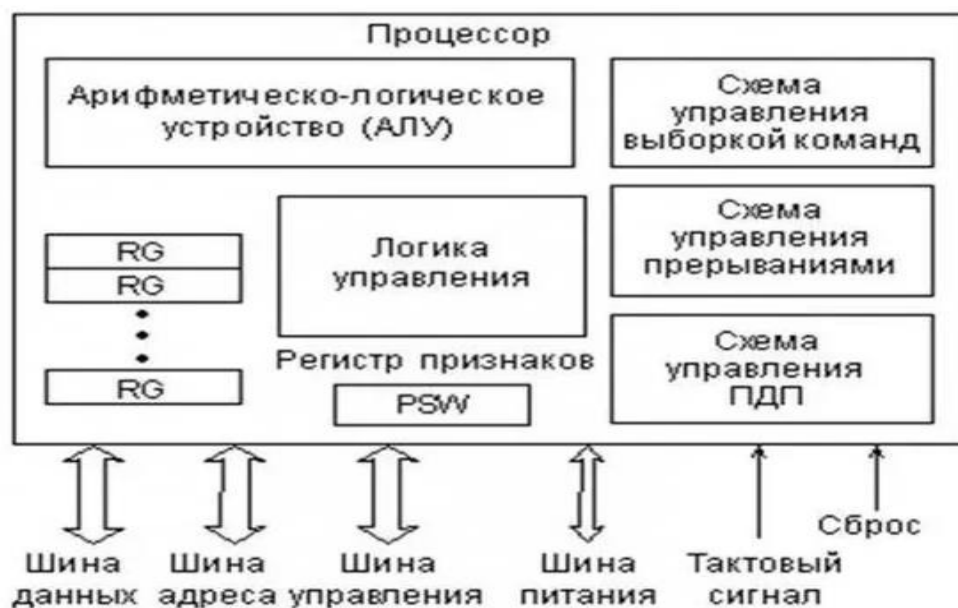


Рисунок 2.8 – внутренняя структура микропроцессора

Адресная шина. У процессоров семейства Pentium (а именно они наиболее распространены в персональных компьютерах) адресная шина 32-разрядная, то есть состоит из 32 параллельных проводников. В зависимости от

того, есть напряжение на какой-то из линий или нет, говорят, что на этой линии выставлена единица или ноль. Комбинация из 32 нулей и единиц образует 32-разрядный адрес, указывающий на одну из ячеек оперативной памяти. К ней и подключается процессор для копирования данных из ячейки в один из своих регистров.

Шина данных. По этой шине происходит копирование данных из оперативной памяти в регистры процессора и обратно. В современных персональных компьютерах шина данных, как правило, 64-разрядная, то есть состоит из 64 линий, по которым за один раз на обработку поступают сразу 8 байтов.

Шина команд . Для того чтобы процессор мог обрабатывать данные, ему нужны команды. Он должен знать, что следует сделать с теми байтами, которые хранятся в его регистрах. Эти команды поступают в процессор тоже из оперативной памяти, но не из тех областей, где хранятся массивы данных, а оттуда, где хранятся программы. Команды тоже представлены в виде байтов. В большинстве современных процессоров шина команд 32-разрядная, хотя существуют 64-разрядные процессоры и даже 128-разрядные.

Видов процессоров существует много, выпускаются они для различных целей и разными производителями. Сегодня ведущими производителями процессоров для компьютеров являются компании Intel3 и AMD4 .

Независимо от производителя, у каждого процессора есть целый ряд важных характеристик: тактовая частота, разрядность обрабатываемых данных, размер кэш-памяти, количество ядер.

Тактовая частота – определяет сколько элементарных операций (тактов) выполняет микропроцессор в одну секунду. Измеряется в гигагерцах (ГГц – GHz). От тактовой частоты в значительной степени зависит быстродействие микропроцессора. Но надо заметить, что утверждение «чем выше тактовая частота, тем "шустрее" процессор» справедливо, если сравнивать между собой поколения CPU одной марки. Сопоставлять по этому показателю процессоры разных производителей нельзя – при одинаковой тактовой частоте они работают с различной скоростью, поскольку на нее влияют в не меньшей степени и другие характеристики. Например, процессоры марки AMD работают на более низких тактовых частотах, чем Intel, но за один такт производят больше действий.

Разрядность (обрабатываемых данных) процессора показывает, сколько бит данных он может принять и обработать в своих регистрах за один раз (за один такт). Очевидно, и эта характеристика процессора влияет на его быстродействие. Первые процессоры x86 были 16-разрядными. Начиная с процессора 80386, они имеют 32-разрядную архитектуру. Подавляющее большинство современных процессоров являются 64-разрядными, но они

полностью поддерживают архитектуру x86. Конечно, для пользователя важно знать, разрядность процессора на его компьютере, так как, например, программное обеспечение, рассчитанное на 64-х разрядный процессор, не может быть установлено на компьютер с 32-х разрядным процессором. Обмен данными внутри процессора происходит в несколько раз быстрее, чем обмен с другими устройствами, например, с оперативной памятью. Для того чтобы уменьшить количество обращений к оперативной памяти, внутри процессора создают буферную область – так называемую кэш-память (англ. cache – тайник, тайный склад).

Кэш-память процессора - это сверхпроизводительная память, откуда процессор получает доступ к обрабатываемым данным. Объем ее очень мал и не позволяет вместить в себя исполняемую программу целиком, поэтому в кэш обычно загружены только часто используемые данные. Разумеется, чем кэш больше, тем к большему объему информации процессор может получить быстрый доступ. Поэтому от величины кэш-памяти зависит скорость исполнения программы.

Большинство современных процессоров оснащены кэш-памятью двух или трех уровней:

1 Кэш-память первого уровня (*L1*) – самый быстрый из всех уровней, выполняется в том же кристалле, что и процессор, за счет чего имеет наименьшее время отклика и работает на скорости близкой к скорости процессора. Имеет объем порядка десятков килобайт. Еще одна функция этого вида памяти – обеспечивать обмен между процессором и вторым уровнем кэш-памяти.

2 Кэш-память второго уровня (*L2*) – имеет больший объем памяти, чем первый. Находится либо в кристалле процессора, либо в том же узле, что и процессор, хотя и выполняется на отдельном кристалле. Одно из предназначений – буфер между вторым и третьим уровнем.

3 Кэш-память третьего уровня (*L3*) – самый медленный из кэшей (но все же значительно быстрее ОЗУ), имеет самый большой объем памяти (может достигать нескольких мегабайт). Выполняют на быстродействующих микросхемах типа *SRAM* и размещают на материнской вблизи процессора.

2.2.4 ОПЕРАТИВНАЯ ПАМЯТЬ

Оперативная память (ОЗУ) – это массив кристаллических ячеек, способных хранить данные. Является достаточно дорогой частью аппаратного обеспечения ПК и оказывает значительное влияние на его производительность и её внешний вид представлен на рисунке 2.9.

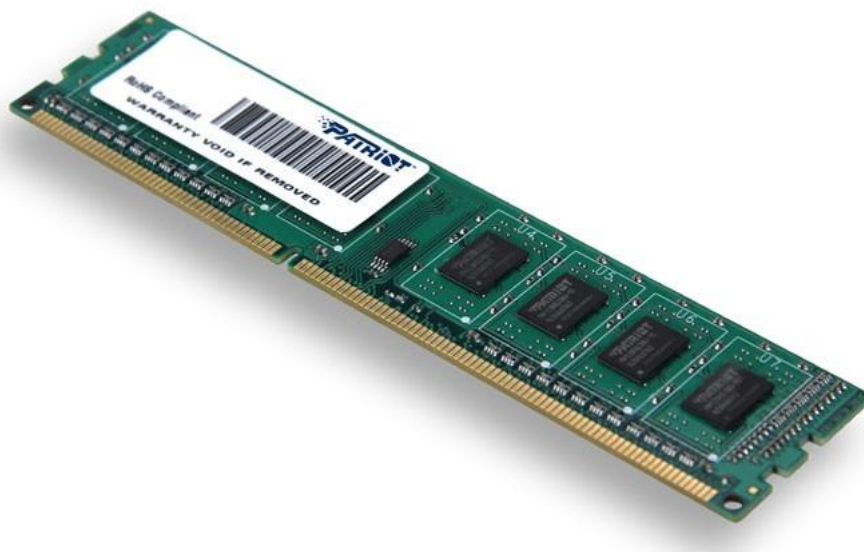


Рисунок 2.9 – модуль оперативной памяти

Из ОЗУ процессор берет программы и исходные данные для обработки, туда же записываются полученные результаты. Но при выключении компьютера ее содержимое стирается.

При обработке данных процессором может произойти обращение к любой ячейке памяти, поэтому ее называют памятью с произвольным доступом (Random Access Memory – RAM). Название «оперативная» отражает высокую скорость ее работы. Наиболее важные характеристики памяти: тип, емкость и скорость передачи данных (пропускная способность). С точки зрения физического принципа различают динамическую память (DRAM) и статическую память (SRAM).

Ячейки динамической памяти (DRAM) можно представить в виде микроконденсаторов, способных накапливать заряд на своих обкладках. Это наиболее распространенный и экономически доступный тип памяти. Недостатки этого типа связаны, во-первых, с тем, что как при заряде, так и при разряде конденсаторов неизбежны переходные процессы, то есть запись данных происходит сравнительно медленно. Второй важный недостаток связан с тем, что заряды ячеек имеют свойство рассеиваться в пространстве, причем весьма быстро. Если оперативную память постоянно не «подзаряжать», утрата данных происходит через несколько сотых секунды. Для борьбы с этим явлением в компьютере происходит постоянная регенерация (освежение, подзарядка) ячеек оперативной памяти. Регенерация осуществляется несколько десятков раз в секунду и вызывает непроизводительный расход ресурсов вычислительной системы. Ячейки статической памяти (SRAM) можно представить как электронные микросхемы – триггеры, состоящие из нескольких

транзисторов. В триггере хранится не заряд, а состояние (включен/выключен), поэтому этот тип памяти обеспечивает более высокое быстродействие, хотя технологически он сложнее и, соответственно дороже. Микросхемы динамической памяти используются в качестве оперативной памяти компьютера. Микросхемы статической памяти используют в качестве кэш-памяти. Оперативная память выпускается в виде модулей памяти (memory module). С точки зрения организации элементов памяти существует два наиболее распространенных вида модулей памяти: SIMM (Single In-line Memory Module – одинарный (односторонний) модуль памяти) - модули памяти с однорядным расположением контактов; DIMM (Dual In-line Memory Module – двойной модуль памяти). Основным отличием DIMM является то, что контакты, расположенные на разных сторонах модуля, являются независимыми, в отличие от SIMM, где симметричные контакты, расположенные на разных сторонах модуля, замкнуты между собой и передают одни и те же сигналы.

2.2.5 ЖЁСТКИЙ ДИСК

Жесткий диск представляет собой один или несколько металлических дисков, покрытых специальным магниточувствительным веществом, которые размещаются на одной оси и заключены в герметизированный корпус из прессованного алюминия. Также содержит двигатель, головку чтения (записи) и управляющую электронику, часть их которой можно рассмотреть на рисунке 2.10.



Рисунок 2.10 – жёсткий диск

К основным числовым параметрам жесткого диска относятся: **емкость, скорость чтения, среднее время доступа, скорость вращения диска, размер кэш-памяти**. Параметр **емкость**, очевидно, определяет, какое количество информации он может хранить. Надо заметить, что потребности в дисковом пространстве растут более, чем на 50% в год. Если операционной системе MS DOS хватало менее 2 МБ на диске, а Windows 95 – порядка 100 МБ, то Windows XP требуется уже около 2 ГБ. А Windows 7 – более 16 ГБ.

От показателя **скорости вращения** диска зависят скорость доступа и чтения данных. Чем выше эта скорость, тем быстрее вращаются магнитные диски внутри корпуса HDD и тем быстрее происходит чтение и запись информации. Чаще всего можно встретить жёсткие диски со скоростью вращения 5400 (диски большого объёма) и 7200 об/мин. Сегодня стандарт – 7200 об/мин. Скорость чтения данных для современных жестких дисков не превышает 50 МБ/с.

Среднее время доступа измеряется в миллисекундах и определяет интервал времени, необходимый для поиска нужных данных, и зависит от скорости вращения диска. Для дисков, вращающихся с частотой 5400 об/мин, среднее время доступа составляет 9-10 мс, для дисков с частотой 7200 об/мин – 7-8 мс.

Кэш-память – быстрая «буферная» память небольшого объема, в которую компьютер помещает наиболее часто используемые данные. Это нужно для того, чтобы информация не считывалась с дисковой пластины при каждом запросе. В результате достигается более высокая скорость обработки данных. Чем больше размер кэша, тем лучше. Распространены винчестеры с объёмом кэша 8, 16, 32 и 64 МБ. Современный винчестер имеет объем от 160 ГБ, буфер – от 8 МБ, время доступа менее 9 мс, скорость интерфейса передачи данных – от 300 МБ/с, скорость вращения – 7200 об/мин у обычных дисков (до 15 000 об/мин у дисков для серверов).

3 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

3.1 ОБЗОР НАИБОЛЕЕ ИСПОЛЬЗУЕМЫХ ПОДХОДОВ В ПРОЕКТИРОВАНИИ ПРОГРАММНЫХ ПРОДУКТОВ

В современной разработке программного обеспечения существует два основных подхода: структурный и объектно-ориентированный.

Структурный. Структурный хорошо работает для систем средней сложности и базируется на принципе функциональной декомпозиции. Основным строительным блоком структурного проектирования является программный модуль - функция, а внимание уделяется, прежде всего вопросам передачи управления и вопросам декомпозиции больших алгоритмов на меньшие. Недостатком такого подхода является то, что система нелегко адаптируется и при увеличении размера приложения сопровождать их становится сложнее.

Объектно-ориентированный. Наиболее современный подход к разработке ПО - это объектно-ориентированный. Здесь в качестве основного строительного блока выступает объект или класс. Объектно-ориентированный подход в области разработки ПО используется потому, что он продемонстрировал свою полезность при построении систем любого размера и сложности в самых разных областях. Кроме того, большинство современных языков программирования, инструментальных средств и операционных систем являются в той или иной мере объектно-ориентированными, и это даёт веские основания судить о мире в терминах объектов. Объектно-ориентированные методы разработки легли в основу идеологии сборки систем из отдельных компонентов.

3.2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

В данном разделе рассмотрим основные концепции в структурном проектировании программных продуктов. Выявим достоинства и недостатки, а также выделим принципиальные отличия от объектно-ориентированного подхода.

3.2.1 НЕДОСТАТКИ ООП

1 В большинстве языков высокого уровня обработка ошибок реализована с помощью механизма исключений. Идея о том, что выполнение кода нужно

прерывать, как только достигнуто некорректное состояние, хороша, но её реализация в виде выброса исключения и перехвата его в другом месте требует от коллектива программистов железной дисциплины. Иначе простая линейная цепочка вычислений превратится в древообразную, а то и в запутанный граф. Функциональная парадигма предоставляет абстракции для простой и прозрачной обработки ошибок.

2 Императивный код подобен тому, как человек представляет себе порядок каких-то действий. Такой подход очень хорош для описания последовательных вычислений, однако в современном мире большинство приложений асинхронные и многопоточные, и здесь уследить за взаимодействием императивного кода, выполняющегося во многих потоках, очень сложно. Наверное, каждый программист сталкивался с ситуациями, когда два потока пытаются изменить одно и то же значение, или же когда один поток ждёт другой поток, а тот, в свою очередь, ждёт первый?

3 Не существует математически строгой, аксиоматизированной и пригодной к повседневному применению теории, описывающей паттерны проектирования и систему типов. Паттерны проектирования — это, конечно, большой шаг в сторону стандартизации способов проектирования, но и они — скорее сборник советов и указаний, нежели строгая теория. Исходя из этого, комбинируя паттерны, вы никогда с полной уверенностью не будете знать, что получите в итоге, если вы не делали этого ранее, что, в свою очередь, ведёт к разного рода «неожиданностям» при разработке.

3.2.2 ДОСТОИНСТВА СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ

1 Структурное программирование позволяет значительно сократить число вариантов построения программы по одной и той же спецификации, что приводит к значительному снижению сложности программы.

2 В структурированных программах логически связанные операторы находятся визуально ближе, а слабо связанные дальше, что позволяет обходиться без блок-схем и других графических форм изображения алгоритмов.

3 Сильно упрощается процесс тестирования и отладки структурированных программ.

4 Более высокую производительность работы за счет того, что действие каждой управляющей структуры хорошо известно и нет необходимости его обдумывать.

5 Ясность и читаемость программ.

6 Более высокую эффективность за счет глобальной оптимизации программы.

3.2.3 ФУНКЦИОНАЛЬНЫЙ ПОДХОД

В функциональном программировании решили подойти с другой стороны. Во-первых, решили отказаться от возможности изменять переменные настолько, насколько это возможно (существуют фундаментальные ограничения, не позволяющие сделать это полностью). Функция в смысле информатики не является функцией в смысле математики: например, функция датчика случайных чисел (на основе физического датчика) не является математической функцией — она не принимает параметров, и потому, с точки зрения математики, должна быть константой, но каждый раз эта функция выдаёт разный результат. Такого в математике не бывает. Если же мы запретим переменным изменяться, то есть уничтожим глобальное изменяемое состояние, то функции станут действительно математическими, и можно будет построить строгую математическую теорию. Такая теория была построена и ныне носит название λ -исчисления. Эта теория описывает типы и полиморфизм.

Математиками-топологами была построена достаточно абстрактная теория категорий, которую они использовали для сокращения доказательств. Со временем теория категорий нашла своё применение во множестве отраслей математики. Информатики выяснили, что эта теория подходит для описания операций с реальными данными, которые не всегда могут быть корректными. Например, понятие монады можно применить для описания вычислений, которые могут привести к ошибке.

Монады бывают двух видов. Первый вид — «чистая» монада, она представляет корректные данные. Второй вид — некорректная монада, она нужна для представления некорректных данных. Над этими двумя типами данных определены операции:

`pure` — функция создания «чистой» монады.

`flatMap` — применение преобразования, которое может привести к некорректному результату. К примеру, операция деления целых чисел будет корректно обрабатывать все данные, кроме деления на 0. В этом случае будет создана некорректная монада и применение к ней операции `flatMap` её не изменит.

`map` — применение к данным операции, которая не может вызвать исключений по своей природе, например, операции сложения.

Важно понимать, что функциональное программирование занимает нишу в середине между низкоуровневыми операциями и очень высокими уровнями абстракции — например, такими, как уровень сервисов или приложений. Исходя из этого был создан объектно-функциональный стиль, позволяющий использовать императивный код для низкоуровневых функций, чистый функциональный код в середине приложения и паттерны проектирования и прочие

ООП-технологии на наивысшем уровне абстракции. Таким образом, функциональный подход лучше применять не для создания замкнутых систем, а скорее для слоёв приложений и описания бизнес-логики. Именно в таком ключе функциональное программирование и применяется на сегодняшний день — построенное над низкоуровневым императивным кодом, внутри объектных абстракций.

3.2.4 МЕТОДЫ СТРУКТУРНОГО ПРОЕКТИРОВАНИЯ

При проектировании модульного ПО используют два подхода: метод нисходящего проектирования и метод восходящего проектирования.

Метод нисходящего проектирования. Он подобен процессу постепенного получения более детального изображения предмета из его общего вида путем применения все более сильного увеличения. В данном методе сначала разрабатываются, кодируются, тестируются и отлаживаются программные модули самого верхнего уровня. При этом, чтобы как можно раньше начать проверку работоспособности системы, вместо модулей нижнего уровня, еще детально не разработанных, следует использовать программные заглушки.

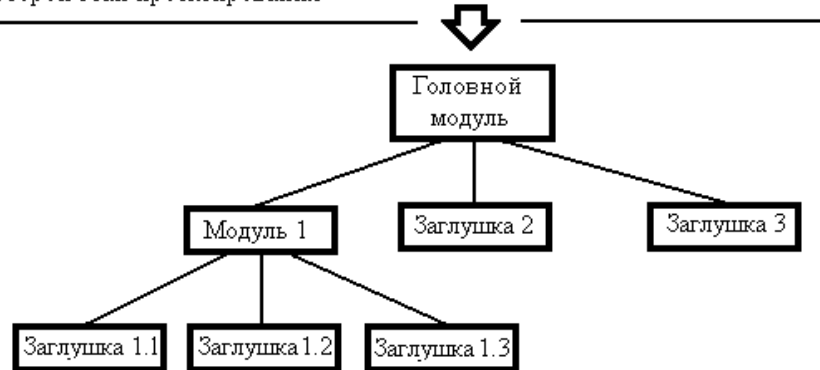
Заглушка – это простой по структуре программный модуль, в котором входные и выходные данные соответствуют замещаемому модулю, но алгоритм обработки данных значительно упрощен. Часто в программе–заглушке помимо описания входных и выходных данных присутствует один оператор печати, сообщаящий о том, что в этом месте программы вызывается заглушка. Простейшая заглушка для ассемблерных программ – это оператор возврата из подпрограмм или блок передачи в выходные ячейки подпрограммы (модуля) заранее подготовленных данных.

Применение метода нисходящего проектирования основано на пошаговом уточнении решения задачи. Начиная с верхних, самых общих шагов, на каждом следующем происходит все большее уточнение функций, выполняемых программой, до полной их реализации. Основное преимущество метода пошагового уточнения состоит в том, что при его использовании особое внимание обращается на проектирование правильной программы соответствующего уровня, а не на детальное понимание сразу всех мелочей задачи. Если первый этап проектирования корректен, то каждый последующий является лишь уточнением предыдущего с небольшими изменениями. Пример нисходящего проектирования приведён на рисунке 3.1:

Первый этап проектирования



Второй этап проектирования



Третий этап проектирования

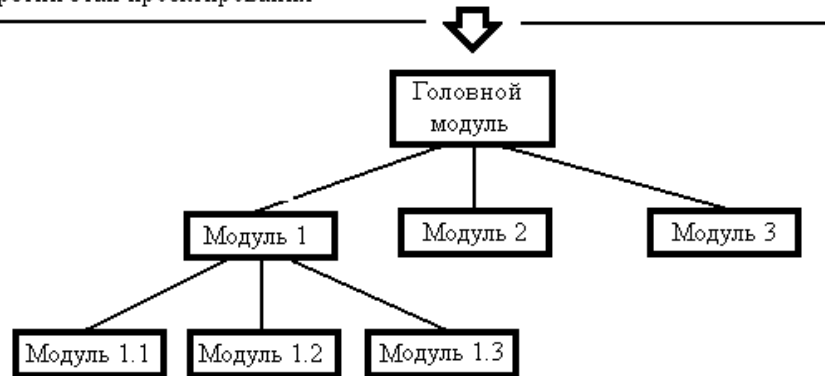


Рисунок 3.1 – пример нисходящего проектирования программного продукта

Метод восходящего проектирования. Отличительной особенностью данного метода является в первую очередь разработка модулей самого нижнего уровня. К таким модулям относятся те, которые уже проверены на работоспособность и далее они включаются в разрабатываемые модули более высокого уровня. Достоинством метода является то, что не требуется писать программы-заглушки, а недостаток заключается в возможности проверки работоспособности модулей верхнего уровня, только после разработки, проверки и отладки модулей нижнего уровня.

3.3 ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

В объектно-ориентированном подходе используется объектная декомпозиция, которая говорит о том, что структура программного продукта описывается объектами и связями между ними. Что касается поведения системы в общем, то она описывается обменом сообщениями между объектами.

Основными элементами объектно-ориентированного подхода являются следующие концепции:

Абстрагирование — выделение наиболее существенных и важных характеристик описываемого объекта для данного программного продукта.

Инкапсуляция — отделение друг от друга отдельных элементов объекта, а также сокрытие реализации от конечного пользователя.

Модульность — возможность декомпозиции ПО на ряд слабо связанных модулей.

Устойчивость — свойство объекта существовать во времени и/или пространстве.

Объектно ориентированный подход проектирования представляет собой циклическое выполнение четырёх основных шагов:

- Определение классов и объектов на определённом уровне абстракции.
- Определение семантики классов.
- Определение (идентификация) связей между классами и объектами.
- Реализация классов.

Также на каждом из повторений уточняется описание классов и перерабатывается проектная документация.

Характеристика объектно-ориентированного подхода. Как правило основными составляющими характеристики объектно-ориентированного подхода являются **объект** и **класс**. **Объект** — это предмет или явление, обладающая четко определяемым поведением. Объект обладает состоянием, поведением и индивидуальностью; **Объект** — это совокупность кода и данных, которые воспринимаются как одно целое. Объект может являться частью приложения, как, например, элемент управления или форма. Приложение в целом также может быть объектом.

Класс — это множество объектов, связанных общностью структуры и поведения. Любой объект является экземпляром класса. Определение классов и объектов - одна из самых сложных задач объектно-ориентированного проектирования. Класс описывает переменные, свойства, процедуры и события объекта. Объекты представляют собой экземпляры классов; после того как класс определен, можно создать любое количество объектов.

Отношения между классами. Самыми распространёнными примерами связей между классами в рамках объектно-ориентированного подхода являются:

1 Агрегация — отношение, когда один объект входит в состав другого, пример которого представлен на рисунке <номер>:

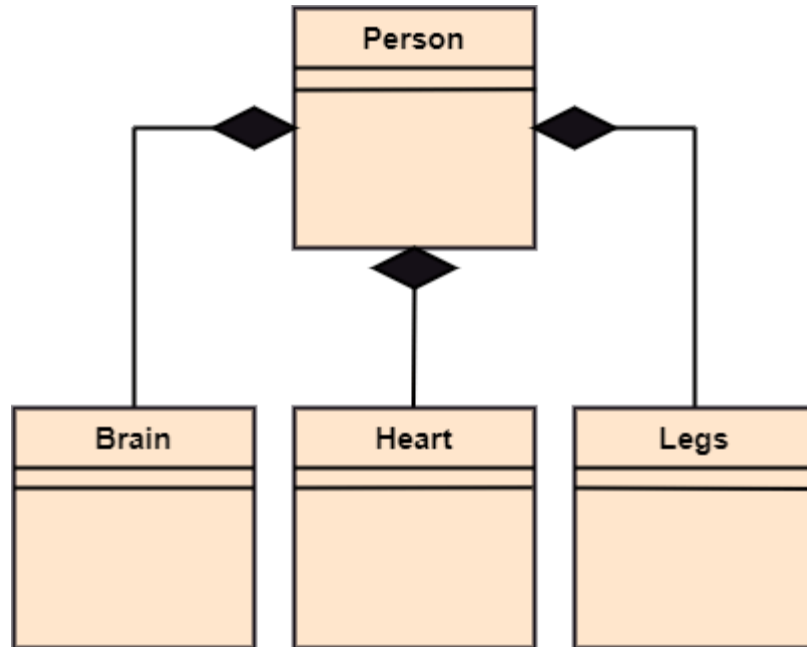


Рисунок 3.2 – агрегация классов, описывающих человеческие органы

2 Ассоциация — отношение между классами, когда каждый ссылается друг на друга, но не носит характер контейнеризации друг друга, которое представлено на рисунке <номер>:

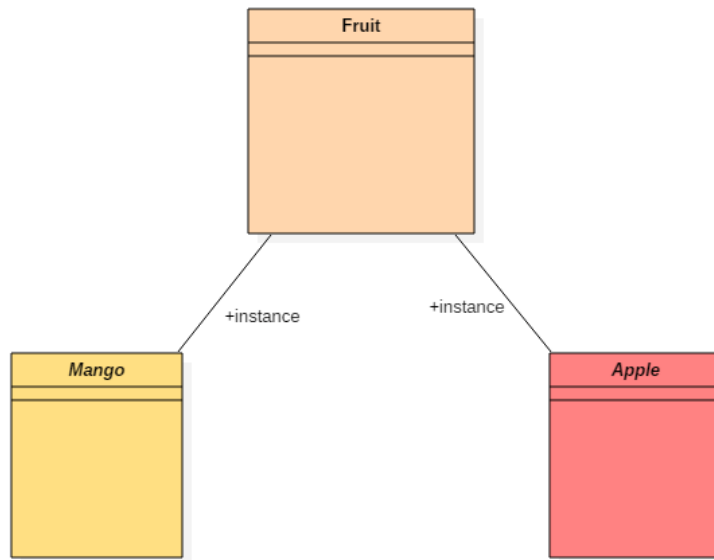


Рисунок 3.3 – ассоциация классов

Наследование — позволяет описать новый класс на основе уже созданного, при этом свойства и методы переходят в новый класс. Пример реализации наследования приведён на рисунке <номер>:

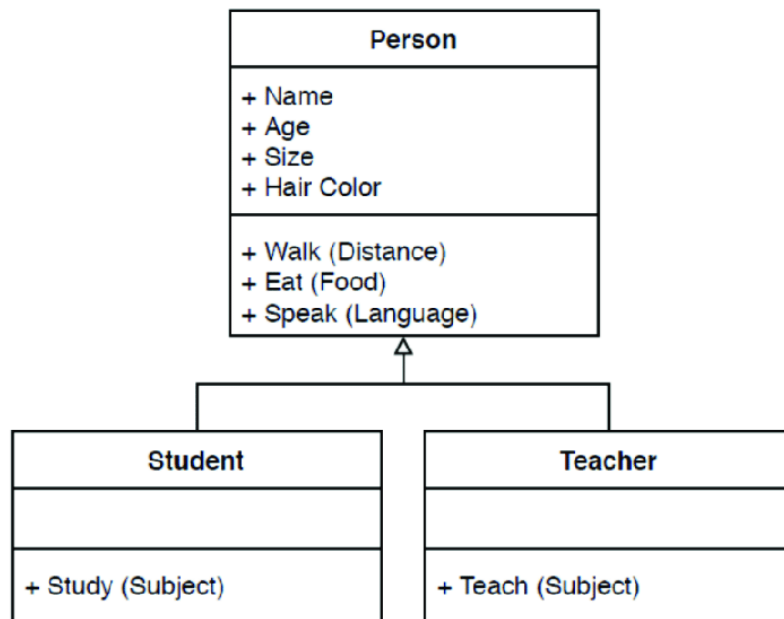


Рисунок 3.4 – наследование классов

- Метаклассы — используется в случаях требуется изменить сам характер системы классов: расширить язык новыми типами классов, изменить стиль взаимодействия между классами и окружением, добавить некоторые дополнительные аспекты, затрагивающие все используемые в приложении классы. Как это реализуется на уровне концепции, представлено на рисунке <номер>:

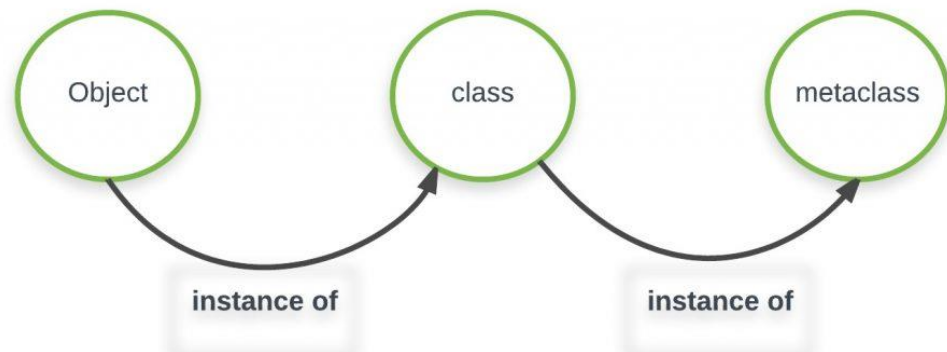


Рисунок 3.5 – концептуальная реализация метакласса

Взяв за основу ранее изложенные требования к программному продукту и рассмотренные подходы к проектированию программного обеспечения возьмём за основу объектно-ориентированный подход проектирования и разработки ПО. Одним из ключевых факторов выбора такого подхода является выбор объектно-ориентированного языка программирования C++ а также реализация графического интерфейса с помощью Qt Creator.

Исходя из изложенных ранее требований к реализуемому программному продукту, определим функциональные возможности программы:

- обработка нажатий клавиш клавиатуры;
- обработка нажатий клавиш компьютерной мыши;
- обработка нажатий на панель навигации приложения;
- вывод графического интерфейса;
- вывод информации на панели навигации приложения;
- вывод графика использования ядер процессора;
- вывод свободного и используемого пространства на твердотельном накопителе;
- вывод свободной и используемой оперативной памяти;
- вывод действующих процессов ОС Windows;
- вывод названия используемого процессора;
- вывод объёма оперативной памяти;
- вывод информации о состоянии аккумулятора;

- вывод информации о видеокарте/видеоадаптере.

На основе вышеперечисленных требований рассмотрим разработанную функциональную схему, изображённую в Приложении В.

Для разрабатываемого программного продукта были изображены основные функциональные элементы, которые включают в себя:

- элементы запуска программного обеспечения;
- обработчики нажатий компьютерной мыши;
- проверка, какой слот был использован.

4 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

4.1 ФАЙЛОВАЯ СТРУКТУРА РАЗРАБАТЫВАЕМОГО ПРОГРАММНОГО ПРОДУКТА

На данном этапе разберём файловую структуру разрабатываемого проекта, выясним за какие функциональные и структурные части отвечают используемые файлы. На рисунке 4.1 представлена файловая структура разрабатываемой программы:

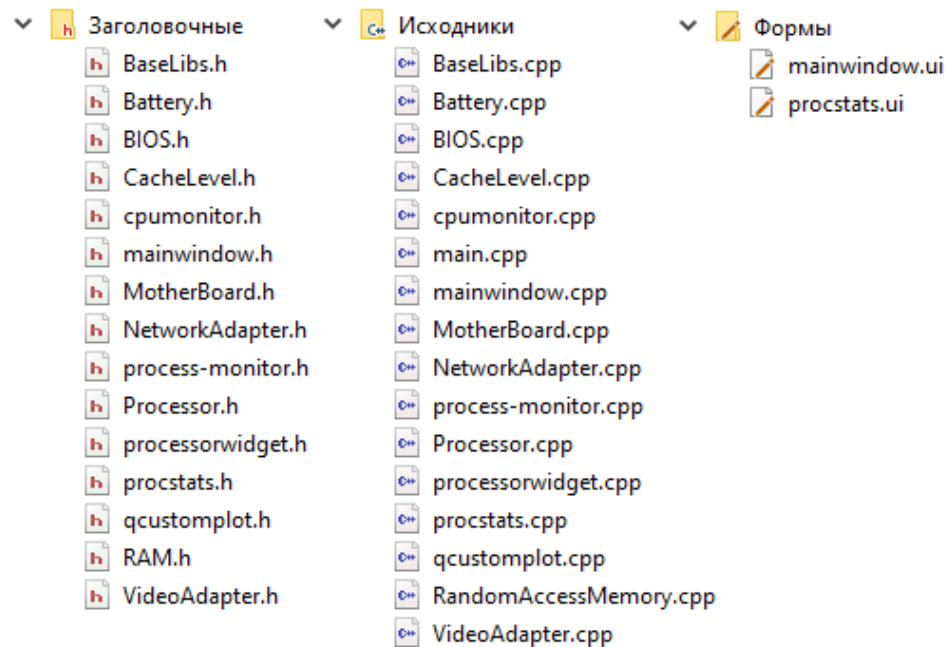


Рисунок 4.1 – файловая структура разрабатываемого продукта

Разрабатываемый проект имеет 3 каталога: Заголовочные, Исходники, Формы.

Заголовочные и Исходники. В данном каталоге расположены все заголовочные файлы программного продукта, которые необходимы в первую очередь для описания будущего функционала программного обеспечения, который будет реализован в файлах каталога Исходники. Также они необходимы для возможности переиспользования уже описанных участков кода в других секциях кода. Например в приложении А, в описании файла BaseLibs.h описаны функции ConvertWCSToMBS и ConvertBSTRToMBS, которые активно используются в каждом классе описания компьютерных комплектующих, при

создании объектов соответствующих классов для конвертации из типа данных BSTR в string. В заголовочный файл как правило входят:

- подключаемые библиотеки;
- описания классов;
- описания функций.

Исходя из изображённой файловой структуры можно сказать, что большую её часть занимают файлы, реализующие объекты классов, содержащие в себе текущую информацию об используемых компьютерных комплектующих пользователя. Инициализация которых происходит следующим образом:

1 Вызывается метод `wmi_initialize(HRESULT* hres, IWbemLocator** ploc, IWbemServices** pSvc)`, который занимается непосредственно подключением к WMI Namespace, предоставляющий доступ к классам Computer System Hardware Classes. Данные классы содержат в себе информацию о компьютерных комплектующих, которая отправляется в соответствующий конструктор класса и преобразуется в соответствии с требованиями, необходимыми для дальнейшей работы продукта.

2 Далее в вызываемом классе используется метод `GetInfo`, который инициализирует и получает объекты `IWbemObject` из которых мы извлекаем допустимую и необходимую информацию о комплектующем.

Отдельно разберём что происходит в файлах `procstats.h` и `qcustomplot.h`. Файл `qcustomplot.h` является дополнительной библиотекой, используемой в Qt Creator для построения графиков функций, диаграмм зависимостей и другого. В разрабатываемом продукте имеется возможность вывода графика использования оперативной памяти конкретным процессом в реальном времени. Соответственно библиотека `qcustomplot.h` используется в файле `procstats.h` для отображения отдельного окна с графиком использования оперативной памяти, внешний вид которого представлен на рисунке 4.2:

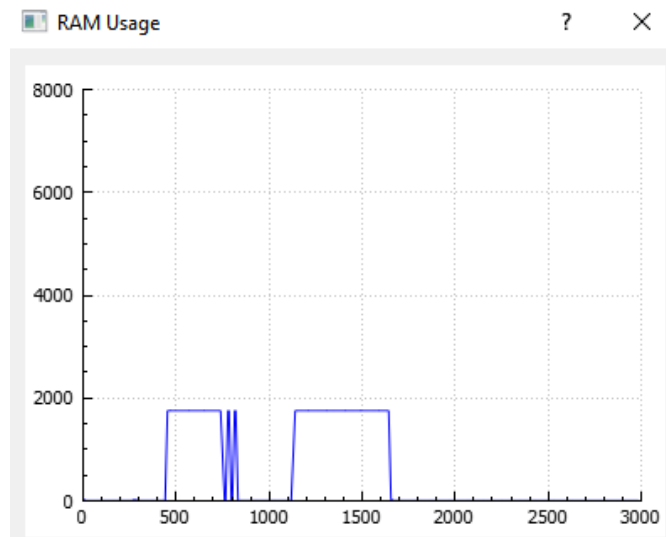


Рисунок 4.2 – график использования RAM для конкретного процесса

Формы. В каталоге **Формы** расположены соответственно файлы `mainwindow.ui` и `procstats.ui`, которые посредством языка XML и соответствующих классов Qt описывают внешний вид программного продукта а также интерфейс взаимодействия с пользователем. Пример внешнего вида формы представлен на рисунке 4.3:

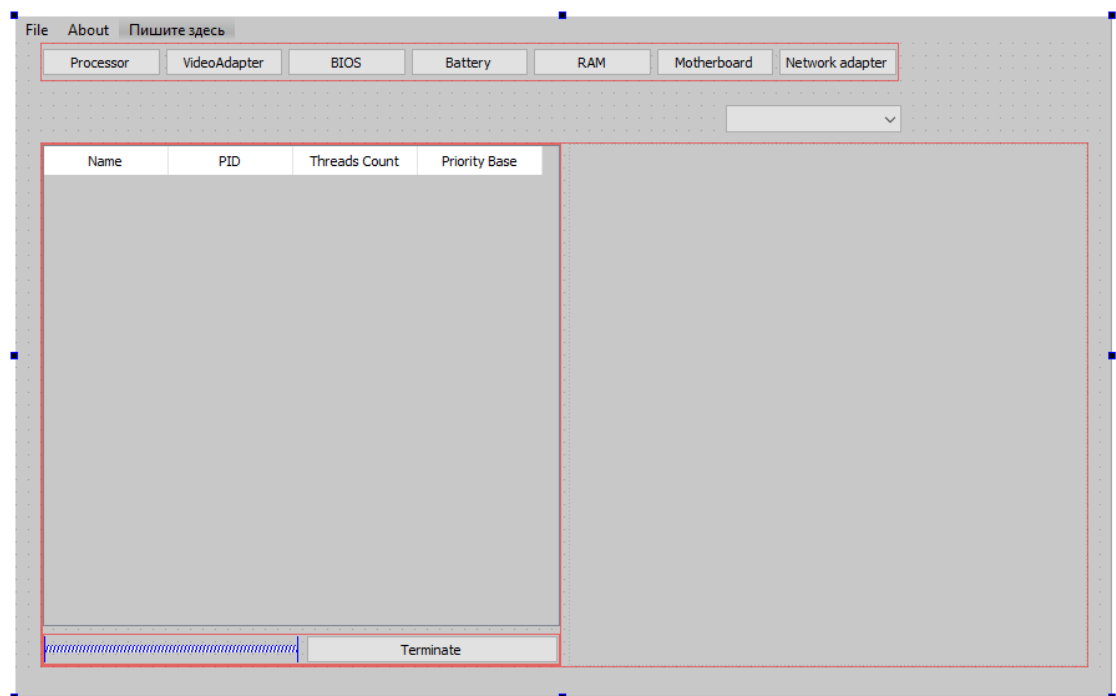


Рисунок 4.3 – пример описания графического интерфейса `mainwindow.ui`

4.2 ДИАГРАММА КЛАССОВ РАЗРАБАТЫВАЕМОГО ПРОГРАММНОГО ПРОДУКТА

Как было упомянуто ранее, файл `mainwindow.ui` описывает графическое представление разрабатываемого продукта, а интерфейс взаимодействия пользователя и приложения обеспечивают соответствующие классы в Qt Creator. Данное программное обеспечение разрабатывается на основе класса `QMainWindow`, который позволяет описывать основное окно взаимодействия с пользователем. На рисунке 4.3 рассмотрим основные используемые компоненты:

- группа `PushButton` для вывода информации о комплектующих;
- выпадающее меню, которым можно выбрать конкретное комплектующее из предложенного списка;
- таблица для вывода информации о процессах в ОС Windows;
- кнопка `Terminate` для закрытия выбранного процесса;
- `infoWidget` для определения пространства, где выводится информация о комплектующем.

Далее рассмотрим структуру взаимодействия класса `MainWindow` с классами описания аппаратных характеристик, представленную на рисунке 4.4:

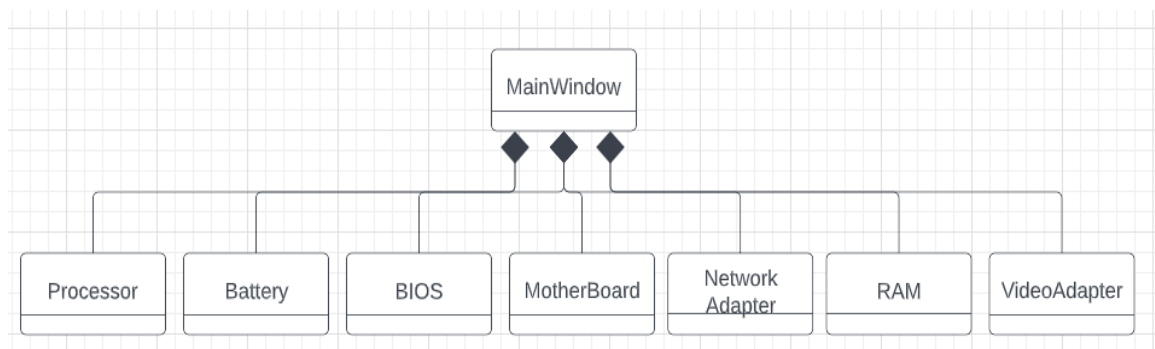


Рисунок 4.4 – диаграмма классов взаимодействия с `MainWindow`

Как видно из рисунка 4.4, для управления работой надо классами аппаратных характеристик используется вид связи Композиция, для которого характерно наличие объектов других классов внутри него, а также ограничение жизненного цикла классом `MainWindow`, что означает, если вызывается деструктор для класса `MainWindow`, а композиционные классы также существуют, то они также будут уничтожены.

Управление методами композиционных классов происходит с помощью ранее описанного механизма слотов, который в своей сущности реализует

паттерн «Подписчик-издатель». То есть при нажатии кнопки, с описанным механизмом слота, генерируется событие onclick() и далее запускается обработчик данного события. Таким же образом работает механизм двойного нажатия на соответствующий столбец процесса, график использования RAM которого будет выведен в диалоговом окне.

4.3 ОПИСАНИЕ БЛОК-СХЕМЫ АЛГОРИТМА ПРИЛОЖЕНИЯ

На рисунке Б.1 изображена блок-схема алгоритма разрабатываемого программного продукта. Рассмотрим подробнее основные элементы, изображённые на схеме.

Запуск приложения. При запуске приложения вызывается конструктор класса главного окна графического интерфейса MainWindow, который проводит процесс инициализации компонентов, содержащихся в данном классе, подразумевающий собой перечень следующих действий:

- инициализация NULL указателей на объекты комплектующих;
- инициализация пользовательского интерфейса;
- инициализация иконок для кнопок;
- создание слотов для отслеживания действий пользователя;
- получение списка процессов методом updateProcesses();
- установка стилей для tableWidget.

Ожидание действий пользователя. Далее программный продукт переходит в режим ожидания срабатывания слота, после которого, в зависимости от того, к какому компоненту пользовательского интерфейса относится сработавший слот. После срабатывания слота проводится проверка на то, к какому из объектов он относится. Алгоритм проверки можно детально рассмотреть на рисунке Б.1. Соответственно, если слот относится к PushButton, далее выполняется поиск и выполнение обработчика, вызванного слота.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта были получены теоретические и практические навыки в разработке программного обеспечения на базе платформы Qt и языке программирования C++. Также были изучены основные аппаратные составляющие компьютера и их аппаратные характеристики. В качестве архитектуры программного продукта были предложены и разработаны функциональные, структурные схемы, а также блок-схемы алгоритмов программного продукта. С помощью платформы Qt и её соответствующих компонентов был разработан графический интерфейс программного обеспечения. На базе WinAPI и средств платформы Qt для языка программирования C++ были разработаны основные функциональные блоки для получения информации о состоянии ОС Windows а также были получены соответствующие аппаратные характеристики персонального компьютера.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

- [1] Алексеев, Е.Р. Программирование на языке C++ в среде Qt Creator/Е.Р. Алексеев. – ALT Linux, 2015. – 448 с.
- [2] Шлее, М. Qt 5.10. Профессиональное программирование на C++. – СПб. : БХВ-Петербург, 2018. – 1072 с.
- [3] 10 секретов «Диспетчера задач» [Электронный ресурс]: MyFreeSoft. – Режим доступа: <https://myfreesoft.ru/10-task-manager-secrets.html>. Дата доступа: 01.12.2022.
- [4] Как использовать диспетчер задач в Windows [Электронный ресурс]: Vynesimozg. – Режим доступа: <https://vynesimozg.com/kak-ispolzovat-dispatcher-zadach-windows-8-ili-10/>. Дата доступа: 01.12.2022.
- [5] Материнская плата. Схема и характеристики. [Электронный ресурс]: 2hpc. – Режим доступа: <https://2hpc.ru/>. Дата доступа: 01.12.2022.
- [6] Микропроцессоры. Основные характеристики. [Электронный ресурс]: bstudy. – Режим доступа: https://bstudy.net/814892/informatika/osnovnye_harakteristiki_mikroprotessorov. Дата доступа: 01.12.2022.
- [7] Внутреннее устройство микропроцессора. [Электронный ресурс]: рх2013. – Режим доступа: <https://apx2013.ucoz.ru/index/0-15>. Дата доступа: 01.12.2022.
- [8] Основные характеристики оперативной памяти. [Электронный ресурс]: Comp-profi. – Режим доступа: <http://comp-profi.com/operativnaya-pamyat-kompyutera-harakteristiki/>. Дата доступа: 01.12.2022.
- [9] Характеристики жёстких дисков. [Электронный ресурс]: Lumpics. – Режим доступа: <https://lumpics.ru/main-hard-disk-specifications/>. Дата доступа: 01.12.2022.
- [10] Побегайло, А.П. Системное программирование в Windows. – СПб. : БХВ-Петербург, 2006. – 1056 с.
- [11] Windows System Programming Fourth Edition – Johnson, M. Hart, ISBN 978-0-321-65774-9 2010.
- [12] Указатель API Windows [Электронный ресурс]: Datasheet / Microsoft. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/apiindex/windows-api-list>. Дата доступа: 01.12.2022.
- [13] Qt official resource [Электронный ресурс]: Datasheet / The QT Company. – Режим доступа: <https://www.qt.io>. Дата доступа: 01.12.2022.
- [14] QtWiki for Beginners [Электронный ресурс]: The QT Company. – Режим доступа: https://wiki.qt.io/Qt_for_Beginners. Дата доступа: 01.12.2022.
- [15] Qt5 Tutorial [Электронный ресурс]: ZetCode. – Режим доступа: <https://zetcode.com/gui/qt5>. Дата доступа: 01.12.2022.

- [16] Head First. Паттерны проектирования / Э. Фримен [и др.] – 2022. – 641 с.
- [17] Структурное проектирование [Электронный ресурс]: Studme. – Режим доступа: https://studme.org/192620/informatika/strukturnoe_proektirovanie. Дата доступа: 01.12.2022.
- [18] Структурное проектирование [Электронный ресурс]: Studfile. – Режим доступа: <https://studfile.net/preview/7209950/page:13/>. Дата доступа: 01.12.2022.
- [19] Методология проектирования программных продуктов [Электронный ресурс]: StudIzba. – Режим доступа: <https://studizba.com/lectures/informatika-i-programmirovaniye/lekicii-po-informatike-i-programmirovaniyu/4303-metodologiya-proektirovaniya-programmnyh-produktov.html>. Дата доступа: 01.12.2022.
- [20] Самые важные аппаратные характеристики компьютера [Электронный ресурс]: Unetway. Режим доступа: <https://studfile.net/preview/7209950/page:13/>. Дата доступа: 01.12.2022.

ПРИЛОЖЕНИЕ А
(обязательное)
ЛИСТИНГ КОДА ПРОГРАММЫ

BaseLibs.cpp

```
#include "BaseLibs.h"

int wmi_initialize(HRESULT* hres, IWbemLocator** pLoc, IWbemServices** pSvc) {
    *hres = CoCreateInstance(
        CLSID_WbemLocator,
        0,
        CLSCTX_INPROC_SERVER,
        IID_IWbemLocator, (LPVOID*)pLoc);

    if (FAILED(*hres))
    {
        cout << "Failed to create IWbemLocator object. "
            << "Err code = 0x"
            << hex << *hres << endl;
        CoUninitialize();
        return 1;
    }

    *hres = (*pLoc)->ConnectServer(
        _bstr_t(L"ROOT\\CIMV2"),
        NULL,
        NULL,
        0,
        NULL,
        0,
        0,
        pSvc
    );

    if (FAILED(*hres))
    {
        cout << "Could not connect. Error code = 0x"
            << hex << hres << endl;
        (*pLoc)->Release();
        CoUninitialize();
        return 1;          // Program has failed.
    }

    *hres = CoSetProxyBlanket(
        *pSvc,          // Indicates the proxy to set
        RPC_C_AUTHN_WINNT,    // RPC_C_AUTHN_XXX
```

```

    RPC_C_AUTHZ_NONE,          // RPC_C_AUTHZ_xxx
    NULL,                      // Server principal name
    RPC_C_AUTHN_LEVEL_CALL,    // RPC_C_AUTHN_LEVEL_xxx
    RPC_C_IMP_LEVEL_IMPERSONATE, // RPC_C_IMP_LEVEL_xxx
    NULL,                      // client identity
    EOAC_NONE                  // proxy capabilities
);

if (FAILED(*hres))
{
    cout << "Could not set proxy blanket. Error code = 0x"
        << hex << *hres << endl;
    (*pSvc)->Release();
    (*pLoc)->Release();
    CoUninitialize();
    return 1;
}
return 0;
}

string ConvertWCSToMBS(const wchar_t* pstr, long wslen)
{
    int len = ::WideCharToMultiByte(CP_ACP, 0, pstr, wslen, NULL, 0, NULL, NULL);

    string dblstr(len, '\0');
    len = ::WideCharToMultiByte(CP_ACP, 0, pstr, wslen, &dblstr[0], len, NULL, NULL);

    return dblstr;
}

string ConvertBSTRToMBS(BSTR bstr)
{
    int wslen = ::SysStringLen(bstr);
    return ConvertWCSToMBS((wchar_t*)bstr, wslen);
}

```

Mainwindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "procstats.h"
#include "BaseLibs.h"
#include "process-monitor.h"

#include <QtWidgets>

```



```
#include <QMessageBox>
#include <QDir>
```

```
void MainWindow::ClearHard(){
    if(this->proc != NULL){
        delete this->proc; this->proc = NULL;
    }
    delete this->battery; this->battery = NULL;
    delete this->bios; this->bios = NULL;
    this->cache.clear();
    delete this->board; this->board = NULL;
    this->nt_adap.clear();
    this->ram.clear();
    this->vid_adapter.clear();
}
```

```
QIcon returnPixmap(const char* path){
    return QIcon(QPixmap(path));
}
```

```
void MainWindow::initializeIcons()
{
    QPixmap pixmap("C:/Users/nikoz/Desktop/img.jpg");
    QIcon ButtonIcon(pixmap);
    ui->procButton-
>setIcon(returnPixmap("C:/Users/nikoz/Desktop/Kursach/pics/processor.png"));
    ui->procButton->setIconSize(QSize(16,16));

    ui->vaButton->setIcon(returnPixmap("C:/Users/nikoz/Desktop/Kursach/pics/video-
adapter.png"));
    ui->vaButton->setIconSize(QSize(16,16));

    ui->ramButton-
>setIcon(returnPixmap("C:/Users/nikoz/Desktop/Kursach/pics/ram.png"));
    ui->ramButton->setIconSize(QSize(16,16));

    ui->naButton-
>setIcon(returnPixmap("C:/Users/nikoz/Desktop/Kursach/pics/network-
adapter.png"));
}
```

```

    ui->naButton->setIconSize(QSize(16,16));

    ui->biosButton-
>setIcon(returnPixmap("C:/Users/nikoz/Desktop/Kursach/pics/bios.png"));
    ui->biosButton->setIconSize(QSize(16,16));

    ui->mothButton-
>setIcon(returnPixmap("C:/Users/nikoz/Desktop/Kursach/pics/motherboard.png"));
    ui->mothButton->setIconSize(QSize(16,16));

    ui->batteryButton-
>setIcon(returnPixmap("C:/Users/nikoz/Desktop/Kursach/pics/battery.png"));
    ui->batteryButton->setIconSize(QSize(16,16));
}

MainWindow::MainWindow(QWidget* parent)
: QMainWindow(parent),
  ui(new Ui::MainWindow)
{
    proc = NULL;
    battery = NULL;
    bios = NULL;
    board = NULL;
    form_layout = new QFormLayout();

    ui->setupUi(this);
    initializeIcons();
    ui->hardChoose->hide();
    updateProcesses(ui);

    ui->tableWidget->setStyleSheet("QTableWidget{ "
                                   "background-color: #C0C0C0;"
                                   "alternate-background-color: #606060;"
                                   "selection-background-color: #282828;"
                                   "}");

    ui->tableWidget->setAlternatingRowColors(true);
    ui->tableWidget->setSelectionMode(QAbstractItemView::SingleSelection);
    ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);

```

```

    ui->tableWidget->setTextElideMode(Qt::ElideRight);
}

void MainWindow::updateProcesses(Ui::MainWindow *ui){
    QList<Process*> list = GetProcessList();
    ui->tableWidget->setRowCount(list.size());
    int i = ui->tableWidget->rowCount() - 1;
    for (QList<Process*>::iterator it = list.begin(); it != list.end(); ++it){
        QTableWidgetItem* item;

        for(int j = 0; j < ui->tableWidget->columnCount(); j++){
            item = new QTableWidgetItem;
            item->setFlags(item->flags() & ~Qt::ItemIsEditable);
            if(j == 0){
                item->setText((*it)->name);
                ui->tableWidget->setItem(i, j, item);
            }
            else if (j == 1){
                item->setText(QString::number((*it)->proc_id));
                ui->tableWidget->setItem(i, j, item);
            }
            else if (j == 2){
                item->setText((*it)->th_count);
                ui->tableWidget->setItem(i, j, item);
            }
            else if (j == 3){
                item->setText((*it)->prior_base);
                ui->tableWidget->setItem(i, j, item);
            }
        }
        i -= 1;
    }
}

MainWindow::~MainWindow()
{
    delete ui;
}

```

```

void MainWindow::clearWidget(){

    while(this->form_layout->rowCount() > 0){

        this->form_layout->removeRow(0);
    }
}

void MainWindow::on_actionOpen_Process_triggered()
{
    QString filter = "Exe file (*.exe)";
    QString file_name = QFileDialog::getOpenFileName(this, "Open a process",
    QDir::homePath(), filter);

    QDesktopServices::openUrl(QUrl("file:///"+file_name, QUrl::TolerantMode));
    updateProcesses(ui);
}

void MainWindow::on_actionInfo_triggered()
{
    QMessageBox msgBox;
    msgBox.setWindowTitle("Task Manager");
    msgBox.setText("This app was builded by Vashkevich Nikolai as course work for
5th term of studying.");
    msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Ok);
    msgBox.exec();
}

void MainWindow::on_actionClose_triggered()
{
    close();
}

void MainWindow::on_pushButton_clicked()
{
    QList<QTableWidgetItem*> items = ui->tableWidget->selectedItems();
    QMessageBox msgBox;
    if(items.size() == 4){
        QTableWidgetItem* item = items.takeAt(1);
        QString text = item->text();
    }
}

```

```

        unsigned int result = text.toUInt();
        killProcessByID(result);
        msgBox.setWindowTitle("Terminate Result");
        msgBox.setText("Process has terminated");
        msgBox.exec();
        updateProcesses(ui);
    }
    else{
        msgBox.setText("You haven't choose any process.");
        msgBox.exec();
    }
}

void MainWindow::on_tableWidget_cellDoubleClicked(int row, int column)
{
    ProcStats window;
    QList<Process*> list = GetProcessList();

    window.setWindowTitle("RAM Usage");
    window.setModal(true);
    window.exec();
}

void MainWindow::on_procButton_clicked()
{
    clearWidget();
    ClearHard();
    ui->hardChoose->clear();
    ui->hardChoose->hide();
    HRESULT hres;
    IWbemLocator *pLoc = NULL;
    IWbemServices *pSvc = NULL;

    wmi_initialize(&hres, &pLoc, &pSvc);

    this->proc = Processor::GetProcessorInfo(&hres, pLoc, pSvc);

    this->timer = new QTimer(this);
    delete ui->infoWidget->layout();
    this->form_layout = CreateLayout(this->proc->ToMap());

```

```

    connect(timer, SIGNAL(timeout()), this, SLOT(UpdateLoad()));
    disconnect(timer, SIGNAL(timeout()), this, SLOT(UpdateLoad()));
    ui->infoWidget->show();
    timer->start(20);
}

QFormLayout* MainWindow::CreateLayout(unordered_map<string, string> map){
    QFormLayout* layout = new QFormLayout(ui->infoWidget);

    for(auto x : map){
        layout->addRow(CreateLabel(QString::fromStdString(x.first), ui->infoWidget),
                       CreateLabel(QString::fromStdString(x.second), ui->infoWidget));
    }

    return layout;
}

QLabel* MainWindow::CreateLabel(const QString name, QWidget* widget){
    QLabel* lab = new QLabel(name, widget);
    lab->setAlignment(Qt::AlignCenter);

    return lab;
}

void MainWindow::UpdateLoad(){
    Sleep(500);
    this->proc->UpdateProcStats();
    delete ui->infoWidget->layout();
    this->form_layout = CreateLayout(this->proc->ToMap());
    ui->infoWidget->show();
}

void MainWindow::on_vaButton_clicked()
{
    clearWidget();
    ClearHard();
    HRESULT hres;
    IWbemLocator *pLoc = NULL;
    IWbemServices *pSvc = NULL;

```

```

wmi_initialize(&hres, &pLoc, &pSvc);
if (ui->hardChoose->count() > 0){
    ui->hardChoose->clear();
}
this->vid_adapter = VideoAdapter::VideoAdapterInfo(&hres, pLoc, pSvc);
for(int i = 0; i < this->vid_adapter.size(); i++){
    ui->hardChoose->addItem(QString::fromStdString(this->vid_adapter[i]-
>name));
}
ui->hardChoose->setCurrentIndex(0);
ui->hardChoose->show();

for(int i = 0; i < this->form_layout->rowCount(); i++){
    this->form_layout->removeRow(0);
}

delete ui->infoWidget->layout();
this->form_layout = CreateLayout(this->vid_adapter[0]->ToMap());

}

void MainWindow::on_hardChoose_activated(int index)
{
    clearWidget();
    delete ui->infoWidget->layout();

    if(nt_adap.size() == 0 && ram.size() == 0){
        this->form_layout = CreateLayout(this->vid_adapter.at(index)->ToMap());
    }
    else if(vid_adapter.size() == 0 && ram.size() == 0){
        this->form_layout = CreateLayout(this->nt_adap.at(index)->ToMap());
    }
    else{
        this->form_layout = CreateLayout(this->ram.at(index)->ToMap());
    }
}

void MainWindow::on_ramButton_clicked()
{

```

```

clearWidget();
ClearHard();
HRESULT hres;
IWbemLocator *pLoc = NULL;
IWbemServices *pSvc = NULL;

wmi_initialize(&hres, &pLoc, &pSvc);

this->ram = RAM::GetRamInfo(&hres, pLoc, pSvc);
for(int i = 0; i < this->form_layout->rowCount(); i++){
    this->form_layout->removeRow(0);
}
delete ui->infoWidget->layout();
ui->hardChoose->clear();

for(int i = 0; i < this->ram.size(); i++){
    ui->hardChoose->addItem(QString::fromStdString(this->ram[i]-
>manufacturer));
}
ui->hardChoose->setCurrentIndex(0);
ui->hardChoose->show();
this->form_layout = CreateLayout(this->ram[0]->ToMap());
}

void MainWindow::on_naButton_clicked()
{
    clearWidget();
    ClearHard();
    HRESULT hres;
    IWbemLocator *pLoc = NULL;
    IWbemServices *pSvc = NULL;

    wmi_initialize(&hres, &pLoc, &pSvc);

    this->nt_adap = NetworkAdapter::GetNetworkAdapterInfo(&hres, pLoc, pSvc);
    ui->infoWidget->clearMask();

    for(int i = 0; i < this->form_layout->rowCount(); i++){
        this->form_layout->removeRow(0);
    }
}

```



```

delete ui->infoWidget->layout();
ui->hardChoose->clear();

for(int i = 0; i < this->nt_adap.size(); i++){
    ui->hardChoose->addItem(QString::fromStdString(this->nt_adap[i]->caption));
}
ui->hardChoose->setCurrentIndex(0);
ui->hardChoose->show();
this->form_layout = CreateLayout(this->nt_adap[0]->ToMap());
}

void MainWindow::on_biosButton_clicked()
{
    clearWidget();
    ClearHard();
    HRESULT hres;
    IWbemLocator *pLoc = NULL;
    IWbemServices *pSvc = NULL;

    wmi_initialize(&hres, &pLoc, &pSvc);

    this->bios = BIOS::GetBiosInfo(&hres, pLoc, pSvc);

    for(int i = 0; i < this->form_layout->rowCount(); i++){
        this->form_layout->removeRow(0);
    }

    delete ui->infoWidget->layout();
    if(!ui->hardChoose->isHidden())
    {
        ui->hardChoose->hide();
    }
    this->form_layout = CreateLayout(this->bios->ToMap());
}

void MainWindow::on_batteryButton_clicked()
{
    clearWidget();
    ClearHard();

```

```

HRESULT hres;
IWbemLocator *pLoc = NULL;
IWbemServices *pSvc = NULL;

wmi_initialize(&hres, &pLoc, &pSvc);

this->battery = Battery::GetBatteryInfo(&hres, pLoc, pSvc);

clearWidget();

delete ui->infoWidget->layout();
if(!ui->hardChoose->isHidden())
{
    ui->hardChoose->hide();
}
this->form_layout = CreateLayout(this->battery->ToMap());
}

void MainWindow::on_mothButton_clicked()
{
    clearWidget();
    ClearHard();
    HRESULT hres;
    IWbemLocator *pLoc = NULL;
    IWbemServices *pSvc = NULL;

    wmi_initialize(&hres, &pLoc, &pSvc);

    this->board = MotherBoard::GetMotherBoardInfo(&hres, pLoc, pSvc);
    for(int i = 0; i < this->form_layout->rowCount(); i++){
        this->form_layout->removeRow(0);
    }
    delete ui->infoWidget->layout();
    if(!ui->hardChoose->isHidden())
    {
        ui->hardChoose->hide();
    }
    this->form_layout = CreateLayout(this->board->ToMap());
}

```

Processor.cpp

```
#include "Processor.h"
```

```
Processor::Processor() {  
    return;  
}
```

```
float Processor::CalculateCPULoad(unsigned long long idleTicks, unsigned long  
long totalTicks)  
{  
    static unsigned long long _previousTotalTicks = 0;  
    static unsigned long long _previousIdleTicks = 0;  
  
    unsigned long long totalTicksSinceLastTime = totalTicks - _previousTotalTicks;  
    unsigned long long idleTicksSinceLastTime = idleTicks - _previousIdleTicks;  
  
    float ret = 1.0f - ((totalTicksSinceLastTime > 0) ? ((float)idleTicksSinceLastTime)  
/ totalTicksSinceLastTime : 0);  
  
    _previousTotalTicks = totalTicks;  
    _previousIdleTicks = idleTicks;  
    return ret;  
}
```

```
unsigned long long Processor::FileTimeToInt64(const FILETIME& ft)  
{  
    return (((unsigned long long)(ft.dwHighDateTime)) << 32) | ((unsigned long  
long)ft.dwLowDateTime);  
}
```

```
float Processor::GetCPULoad()  
{  
    FILETIME idleTime, kernelTime, userTime;  
    return GetSystemTimes(&idleTime, &kernelTime, &userTime) ?  
CalculateCPULoad(FileTimeToInt64(idleTime), FileTimeToInt64(kernelTime) +  
FileTimeToInt64(userTime)) : -1.0f;  
}
```

```

string Processor::GetArchitecture(UINT arch)
{
    switch (arch)
    {
        case 0: return "x86";
        case 1: return "MIPS";
        case 2: return "Alpha";
        case 3: return "PowerPC";
        case 5: return "ARM";
        case 6: return "IA-64 (Itanium-based systems)";
        case 9: return "x64";
        case 12: return "ARM64";
        default: return "Unknown";
    }
}

string Processor::GetAvailableType(UINT type)
{
    switch(type)
    {
        case 1: return "Other";
        case 2: return "Unknown";
        case 3: return "Running / Full Power";
        case 4: return "Warning";
        case 5: return "In Test";
        case 6: return "Not Applicable";
        case 7: return "Power Off";
        case 8: return "Off Line";
        case 9: return "Off Duty";
        case 10: return "Degraded";
        case 11: return "Not Installed";
        case 12: return "Install Error";
        case 13: return "Power Save - Unknown";
        case 14: return "Power Save - Low Power Mode";
        case 15: return "Power Save - Standby";
        case 16: return "Power Cycle";
        case 17: return "Power Save - Warning";
        case 18: return "Paused";
        case 19: return "Not Ready";
        case 20: return "Not Configured";
    }
}

```

```

        case 21: return "Quiesced";
        case 22: return "Other";
        default: return "Unknown";
    }
}

string Processor::GetProcessorType(UINT type)
{
    switch(type)
    {
        case 1: return "Other";
        case 2: return "Unknown";
        case 3: return "Central Processor";
        case 4: return "Math Processor";
        case 5: return "DSP Processor";
        case 6: return "Video Processor";
    }
}

void Processor::SetCache(vector<CacheLevel> cache)
{
    this->cache = cache;
}

```

```

Processor::Processor(UINT arch, UINT avail_type, float clock_speed, float voltage,
    UINT data_width, BSTR name, UINT cores, UINT en_cores, UINT proc_type,
    UINT t_count, float l_perc, UINT max_speed, float max_voltage)
{
    this->architecture = GetArchitecture(arch);
    this->availability = GetAvailableType(avail_type);
    this->current_speed = clock_speed;
    this->current_voltage = voltage;
    this->data_width = data_width;
    this->name = ConvertBSTRToMBS(name);
    this->cores_num = cores;
    this->en_cores_num = en_cores;
    this->proc_type = GetProcessorType(proc_type);
    this->thread_count = t_count;
    this->load_percentage = l_perc;
}

```

```

    this->max_speed = max_speed;
    this->max_voltage = max_voltage;
}

Processor* Processor::GetProcessorInfo(HRESULT* hres, IWbemLocator* pLoc,
IWbemServices* pSvc)
{
    IEnumWbemClassObject* enumerator = NULL;
    BSTR bstr_wql = SysAllocString(L"WQL");
    BSTR bstr_sql = SysAllocString(L"SELECT * FROM Win32_Processor");

    *hres = pSvc->ExecQuery(
        SysAllocString(L"WQL"),
        SysAllocString(L"SELECT * FROM Win32_Processor"),
        WBEM_FLAG_FORWARD_ONLY |
WBEM_FLAG_RETURN_IMMEDIATELY,
        NULL,
        &enumerator);

    if (FAILED(*hres)) {
        cout << "Query for Win32_BIOS Error code = 0x" << hex << hres << endl;
        pSvc->Release();
        pLoc->Release();
        CoUninitialize();
    }
    SysFreeString(bstr_wql);
    SysFreeString(bstr_sql);

    IWbemClassObject* storageWbemObject = NULL;
    ULONG uReturn = 0;

    Processor* proc = NULL;
    while (enumerator)
    {
        HRESULT hr = enumerator->Next(WBEM_INFINITE, 1,
&storageWbemObject, &uReturn);
        if (0 == uReturn)
        {
            break;
        }
    }
}

```

```

VARIANT* chars = new VARIANT[12];

storageWbemObject->Get(L"Architecture", 0, &chars[0], 0, 0);
storageWbemObject->Get(L"Availability", 0, &chars[1], 0, 0);
storageWbemObject->Get(L"MaxClockSpeed", 0, &chars[3], 0, 0);
storageWbemObject->Get(L"DataWidth", 0, &chars[4], 0, 0);
storageWbemObject->Get(L"Name", 0, &chars[5], 0, 0);
storageWbemObject->Get(L"NumberOfCores", 0, &chars[6], 0, 0);
storageWbemObject->Get(L"NumberOfEnabledCores", 0, &chars[7], 0, 0);
storageWbemObject->Get(L"ProcessorType", 0, &chars[8], 0, 0);
storageWbemObject->Get(L"ThreadCount", 0, &chars[9], 0, 0);
storageWbemObject->Get(L"NumberOfEnabledCores", 0, &chars[10], 0, 0);
storageWbemObject->Get(L"CurrentVoltage", 0, &chars[11], 0, 0);

float cpu_load = Processor::GetCPULoad();
proc = new Processor(chars[0].uintVal, chars[1].uintVal, chars[3].uintVal *
cpu_load,
    chars[11].llVal * cpu_load, chars[4].uintVal, chars[5].bstrVal,
chars[6].uintVal, chars[7].uintVal,
    chars[8].uintVal, chars[9].uintVal, cpu_load, chars[3].uintVal,
chars[11].llVal);

    storageWbemObject->Release();
    delete[] chars;
}

return proc;
}

void Processor::UpdateProcStats(){
    float cpu_load = Processor::GetCPULoad();
    this->current_speed = cpu_load * this->max_speed;
    this->load_percentage = cpu_load;
    this->current_voltage = cpu_load * this->max_voltage;
}

unordered_map<string, string> Processor::ToMap(){

    unordered_map<string, string> map;

```

```
map["Name"] = this->name;
map["Cores Num"] = std::to_string(this->cores_num);
map["Current Voltage"] = std::to_string(this->current_voltage);
map["Max Speed"] = std::to_string(this->max_speed);
map["Processor Type"] = this->proc_type;
map["Cpu Status"] = this->cpu_status;
map["Data Width"] = std::to_string(this->data_width);
map["Max Voltage"] = std::to_string(this->max_voltage);
map["Architecture"] = this->architecture;
map["Availability"] = this->availability;
map["Enabled Cores"] = std::to_string(this->en_cores_num);
map["Thread Count"] = std::to_string(this->thread_count);
map["Current Speed"] = std::to_string(this->current_speed);

return map;
}
```


ПРИЛОЖЕНИЕ Б
(обязательное)
БЛОК-СХЕМА АЛГОРИТМА ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ В
(обязательное)
ФУНКЦИОНАЛЬНАЯ СХЕМА ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ Г
(обязательное)
ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ Д
(обязательное)
ВЕДОМОСТЬ