**Федеральное агентство связи**

**ордена Трудового Красного Знамени**

**федеральное государственное бюджетное**

**учреждение высшего образования**

**«Московский Технический Университет Связи и Информатики»**

Кафедра «Информатика»

**КУРСОВАЯ РАБОТА**

на тему: «Операционные системы»

Выполнил: студент группы БСТ2001

Савкин Д.И.

Проверил: доцент кафедры «Информатика»

Гуриков С.Р.

Москва 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4

ТЕХНИЧЕСКОЕ ЗАДАНИЕ 7

1.1 Основания для разработки 7

1.2 Назначение разработки 7

1.3 Требования к программе 7

1.3.1 Требования к функциональным характеристикам 7

1.3.2 Требования к надёжности 7

1.3.3 Требования к составу и параметрам технических средств 8

1.3.4 Требования к информационной программной совместимости 8

1.3.5 Требования к транспортированию и хранению 8

1.3.6 Требования к программной документации 8

1.3.7 Стадии и этапы разработки 8

ГЛАВА 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 10

1.1 Введение 10

1.2 Основные компоненты и их взаимодействие 11

1.3 Архитектуры ядер операционных систем 12

1.4 Ядро операционной системы Windows 13

ГЛАВА 2 ПРАКТИЧЕСКАЯ ЧАСТЬ 27

2.1 Разработка программы (разработка электронного конспекта) 27

2.2 Разработка программы (разработка тестовой части) 29

2.3 Дополнительные возможности программы 40

ЗАКЛЮЧЕНИЕ 55

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 56

ПРИЛОЖЕНИЕ А 57

ПРИЛОЖЕНИЕ Б 62

# ВВЕДЕНИЕ

**Актуальность темы.** Понимание работы операционной системы изнутри является критически важным атрибутом для программистов системного и прикладного уровней. Благодаря этим знаниям разработчики могут писать более производительное и менее ресурсоёмкое программное обеспечение.

В связи с высокими темпами повышения мощностей современных электронно-вычислительных машин, многие программисты прибегают к использованию абстракций в виде высокоуровневых языков программирования. Безусловно, это позволяет не беспокоиться об архитектуре устройства, под которое ведётся разработка. Однако в лучшем случае такой подход к написанию программ может привести к слишком долгому выполнению исполняемого кода. В худшем — программа не запустится в принципе. Понимание работы операционной системы изнутри является критически важным атрибутом для программистов системного и прикладного уровней. Благодаря этим знаниям разработчики могут писать более производительное и менее ресурсоёмкое программное обеспечение.

Аппаратная составляющая устройств меняется не так кардинально, как это может показаться на первый взгляд. Базовые принципы построения ЭВМ до сих пор остаются актуальными, ведь меняется мощность компонентов, а не их архитектура в целом. На сегодняшний момент мы имеем огромное разнообразие электронно-вычислительных устройств, и их архитектура, охватывающая все уровни, очень сложная. Однако стоит заметить, что никакое прикладное приложение не может существовать без фундамента под названием операционнаясистема.

Разработка операционных систем является одной из наисложнейших задач. Это очень долгий и тяжёлый процесс, требующий глубокого понимания архитектуры устройства, под которое ведётся разработка. Разработка операционной системы невозможна без знания принципов её построения, ведь любая ОС является связующим звеном между аппаратным и прикладным уровнем. Стоить отметить, что на текущий момент времени разработка операционных систем мало пользуется спросом на рынке. Это происходит из-за того, что написание таких систем экономически невыгодное решение, с учётом того, что давно существует относительно стабильные продукты, установка и эксплуатация которых довольно проста и не требует специальных навыков и знаний. Современные программисты легко пользуются абстракциями, которые были придуманы уже очень давно, однако это не освобождает их от необходимого знания архитектуры, на которой они работают. Такие знания помогают лучше разбираться в происходящих процессах и способны сэкономить большое количество времени при отладке своих разработок.

**Целью** данной научно-исследовательской работы является изучение аппаратной архитектуры ЭВМ и принципов разработки операционных систем.

Чтобы достичь поставленной цели, необходимо решить следующие задачи:

1. проанализировать теоретический материал по теме исследования;
2. разработать программный проект, содержащий электронное пособие и текстовую программу по вышеуказанной теме;
3. оформить код всех вопросов в виде функций, возвращающих результат;
4. создать DLL-библиотеку, содержащую код всех вопросов из предыдущего пункта;
5. разработать систему авторизации;
6. организовать вывод результатов тестирования в одномерный массив с выводом в DataGridView;
7. организовать вывод результатов тестирования в базу Microsoft Access;
8. организовать вывод результатов в тестовый документ;
9. организовать вывод результатов тестирования в документ Microsoft Word;
10. организовать вывод результатов тестирования в лист Microsoft Excel;
11. организовать вывод результатов тестирования в одномерный массив;
12. записать макрос, форматирующий лист в Microsoft Excel;
13. организовать краткий вывод результатов в виде диаграммы;
14. разработать графическую заставку.

**Объектом** исследования, проведённого в рамках данной работы, являются операционные системы.

**Предметом** исследования является изучение фундаментальных принципов построения и разработки операционных систем.

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

## 1.1 Основания для разработки

Основанием для разработки является задание, выполненное в соответствии с заданием, полученным от кафедры «Информатика» Московского Технического Университета Связи и Информатики, и утверждённое научным руководителем доцентом кафедры «Информатика» к.п.н. Гуриковым С.Р. Дата утверждения: 02.10.2020.

## 1.2 Назначение разработки

Программный продукт предназначен для ознакомления с основными теоретическими положениями по теме курсовой работы и проверке знаний пользователя с помощью тестовой программы.

## 1.3 Требования к программе

### 1.3.1 Требования к функциональным характеристикам

Разработанный программный продукт должен обеспечить выполнение следующих функций:

* возможность ознакомления с теоретическим материалом по теме курсовой работы,
* возможность вывода результатов исследования для пользователя.

### 1.3.2 Требования к надёжности

Разрабатываемое программное обеспечение должно иметь:

* защиту пользовательских данных от несанкционированного доступа;
* устойчивую работу в соответствии с алгоритмом программы;
* выдачу сообщений об ошибках;
* поддержку диалогового режима в рамках предоставляемых пользователю возможностей.

### 1.3.3 Требования к составу и параметрам технических средств

Процессор с тактовой частотой не менее 2,33 ГГц для настольных PC и ноутбуков или процессор Intel® Atom™ с тактовой частотой не менее 1,6 ГГц для нетбуков. Оперативная память: 1024 Мб. Графическая память: 128 Мб. Дисковое пространство: 50 Мб. Операционная система: Windows XP/Vista/7/8/10

### 1.3.4 Требования к информационной программной совместимости

Программа должна легко устанавливаться, функционировать и корректно работать, при наличии операционной системы семейства Microsoft Windows.

### 1.3.5 Требования к транспортированию и хранению

Программа поставляется на USB-флэш-накопителе. Программная документация поставляется в электронном или печатном виде.

### 1.3.6 Требования к программной документации

В ходе разработки программы должны быть подготовлены:

* исходный код программы,
* описание программы,
* методика испытаний,
* руководство пользователя.

### 1.3.7 Стадии и этапы разработки

Стадии разработки представлены в таблице 1.

Таблица 1 — Стадии разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Номер этапа | Название этапа | Срок выполнения | Отчётность |
| 1 | Утверждение темы | До 02.10.2020 | Выбранная тема утверждена научным руководителем 04.10.2020 |
| 2 | Написание введения | До 12.10.2020 | Оценено научным руководителем 18.10.2020 |
| 3 | Написание технического задания и фрагмента теоретической части | До 29.10.2020 | Оценено научным руководителем 01.11.2020 |
| 4 | Корректировки в теоретической части текущей работы | До 20.11.2020 | Оценено научным руководителем 04.12.2020 |
| 5 | Разработка программного обеспечения | До 23.12.2020 | Сдано 23.12.2020 |
| 6 | Сдача пояснительной записки и исходного кода программы | До 23.12.2020 | Сдано 23.12.2020 |
| 7 | Защита курсовой работы | До 23.12.2020 | Сдано 23.12.2020 |

# ГЛАВА 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 1.1 Введение

**Операционная система** — фундаментальная часть любой электронно-вычислительный машины (далее — ЭВМ), представляющая собой комплекс взаимосвязанных программ, который управляет работой всех устройств компьютера, загрузкой в память и выполнением прикладных программ, обеспечивает интерфейс пользователя. На текущий момент времени существует огромное разнообразие операционных систем, используемых для решения самых разнообразных задач. Операционная система — первая программа, которая загружается в компьютер. В данной работе будет глубоко изучена операционная система Microsoft Windows.

Изучение фундаментальных принципов, заложенных в основу Windows, было бы бессмысленным без понимания алгоритма запуска системы на аппаратном уровне ЭВМ. Во времена, когда были созданы первые работающие прототипы современных ЭВМ, не существовало никаких операционных систем. Все программы выполнялись машиной напрямую. Со временем, когда программы становились сложнее и больше, было абсолютно неудобно тестировать их и выяснять причину ошибки, которая могла возникнуть абсолютно в любом месте. Было решено придумать абстракцию, которая бы позволяла писать программы, не зависимо от аппаратной архитектуры. Этой абстракцией, как уже отмечалось ранее, является операционная система. При запуске ЭВМ происходит ряд действий, который в результате приводит к запуску операционной системы. Рассмотрим основные компоненты аппаратной составляющей и все действия, совершаемые ими, более подробно.

## 1.2 Основные компоненты и их взаимодействие

Основными компонентами аппаратной составляющей любой ЭВМ являются: центральное процессорное устройство, оперативное запоминающее устройство и базовая система ввода/вывода.

1. **Центральное процессорное устройство** (Central Processing Unit,  
   далее — **ЦПУ, CPU**) — электронный блок либо интегральная схема, исполняющая машинные инструкции (код программ), главная часть аппаратного обеспечения компьютера или программируемого логического контроллера.
2. **Оперативное запоминающее устройство** (Random Access Memory, далее — **RAM**) — один из видов памяти компьютера, позволяющий единовременно получить доступ к любой ячейки по её адресу на чтение или запись.
3. **BIOS** (Basic I/O System) — набор микропрограмм, реализующих  
   API (Application Programming Interface) для работы с аппаратурой компьютера и подключёнными к нему устройствами.

При нажатии кнопки питания на устройстве происходит начальная инициализация устройства. Эта инициализация представляет собой серию диагностик, называемых POST — Power-On Self-Test. В процессе диагностики BIOS проверяет физическое наличие необходимых компонентов (CPU, RAM и так далее), определяет носитель, с которого будет загружена операционная система. Чтобы диск считался загрузочным, он должен иметь специальную сигнатуру (так называемую Master Boot Signature, главную загрузочную запись). Master Boot Signature — код и данные, необходимые для последующей загрузки операционной системы. Главная загрузочная запись имеет размер в 512 байт и обязана заканчиваться 0x55 и 0xAA байтами соответственно. BIOS загружает (копирует) данную сигнатуру в RAM по адресу 0x7c00 и передаёт ей дальнейшее управление. Обычно MBR представляет собой набор инструкций, который занимается следующей стадией запуска ОС — загрузкой **ядра**.

## 1.3 Архитектуры ядер операционных систем

**Ядро** (англ. kernel) — центральная часть операционной системы, обеспечивающая приложениям координированный доступ к ресурсам компьютера, таким как процессорное время, память, внешнее аппаратное обеспечение, внешнее устройство ввода и вывода информации. Очевидно, что никакая операционная система не может работать без ядра. На данный момент существую различные типы архитектур ядер.

**Монолитное ядро** предоставляет единый набор абстракций для взаимодействия с компонентами. Все компоненты работают в едином адресном пространстве, что позволяет наилегчайшим образом вести разработку новых модулей (компонентов), однако требует от программистов написания стабильного и качественного кода, так как сбой в одном из компонентов может нарушить работу всей системы в целом. **Модульное ядро** представляет собой современную улучшенную версию монолитного ядра. В таком типе ядра компоненты, которые не нужны в данный момент времени, не работают, а подгружаются по мере необходимости. **Микроядро** предоставляет только элементарные функции управления процессами и минимальный набор абстракций для работы с оборудованием. Большая часть работы осуществляется с помощью специальных пользовательских процессов, называемых сервисами. Более подробная информация о сущности процессов изложена в соответствующем разделе. **Экзоядро** — ядро операционной системы, которое предоставляет API для межпроцессорного взаимодействия, а так же безопасной работы с ресурсами компьютера. **Наноядро** — ядро операционной системы, которое обрабатывает лишь аппаратные прерывания (более подробно о них рассказано в соответствующем разделе), а всю остальную работу перекладывает на программный уровень. **Гибридное ядро** представляет собой модифицированную версию микроядра, объединяющую достоинства и недостатки монолитного и микроядра. Ядра операционных систем Windows NT построены на архитектуре гибридного ядра.

## 1.4 Ядро операционной системы Windows

Общая схема архитектуры гибридного ядра семейства операционных систем Windows представлена на рисунке 1.

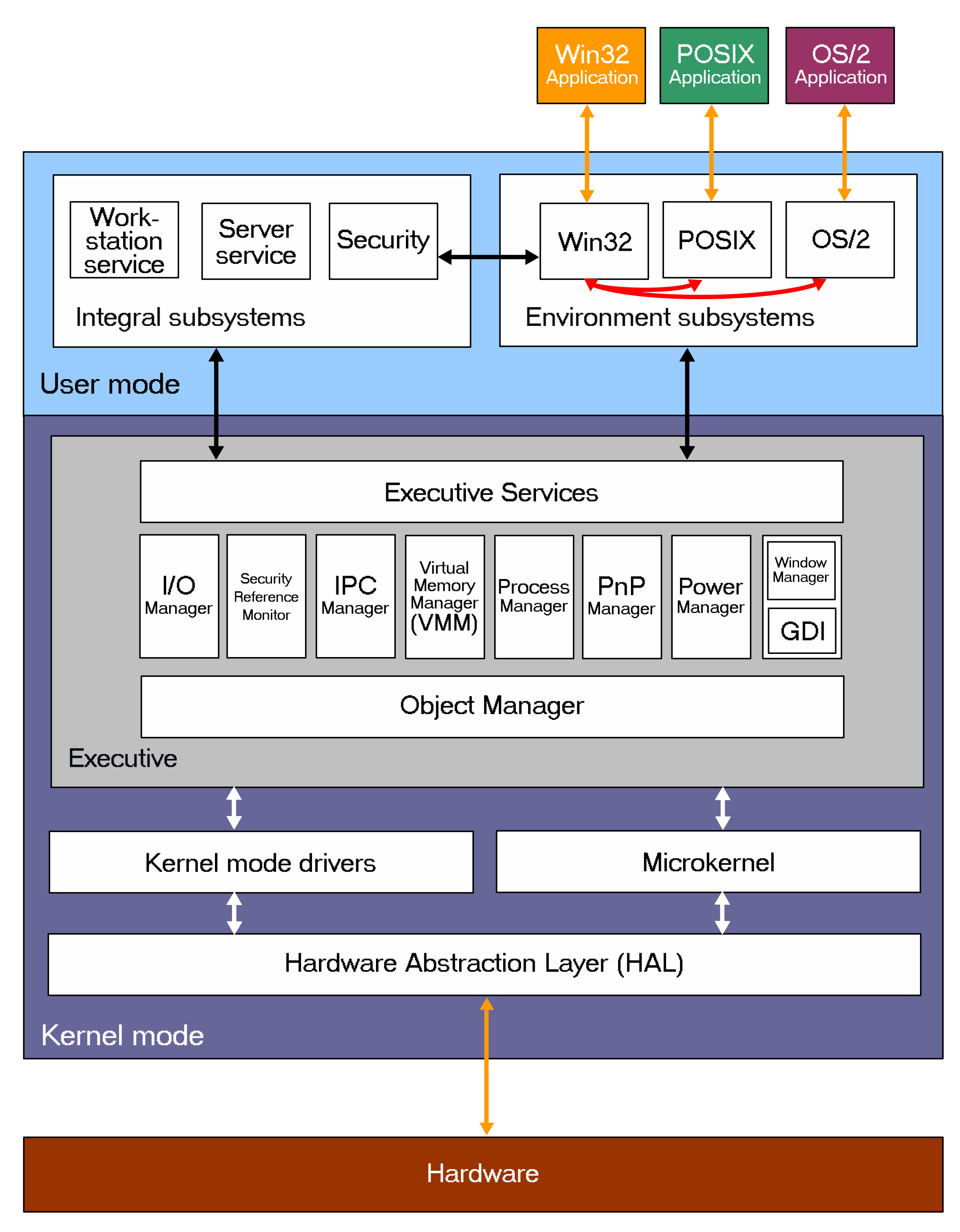


Рисунок 1 — Архитектура ядра Windows NT

Любое процессорное устройство имеет несколько режимов работы, которые определяют набор выполняемых инструкций и поведение при выполнении тех или иных команд. Для упрощения, будем считать, что процессор работает в двух режимах: пользовательском (user mode) и привилегированном (kernel mode). Привилегированный режим доступен только ядру в виду обеспечения безопасности ЭВМ. Очевидно, что выполнение любого программного обеспечения может завершиться с ошибкой, которая способна привести к различным негативным последствиям. Чтобы минимизировать вероятность полного отказа системы в результате появления таких ошибок, любое прикладное обеспечение запускается в пользовательском режиме, который изолирует программы, «помещая» их в «песочницу». Например, драйверы, о сущности которых будет рассказано позже, работают в режиме ядра, ибо являются его частью. В виду этого разработчикам драйверов следует тщательно тестировать свои программные решения, ведь в таком случае ошибка может привести к полному отказу системы.

**Процесс** — абстракция, описывающая выполняющуюся программу, представляющая собой экземпляр выполняемой программы, включая текущее состояние процессора. Концептуально, у каждого процесса есть свой виртуальный CPU. Однако очевидно, что процессов может быть большое количество, а физически процессор один. Чтобы решить данную проблему, все процессы работают последовательно определённый промежуток времени. Такая модель распределения процессорными ресурсами именуется **многозадачным режимом работы**. Процессор буквально переключается между существующими процессами, создавая ощущение (псевдо) параллельности работы всех запущенных программ. На рисунке 2 показан компьютер, работающий в многозадачном режиме и имеющий в памяти четыре программы.

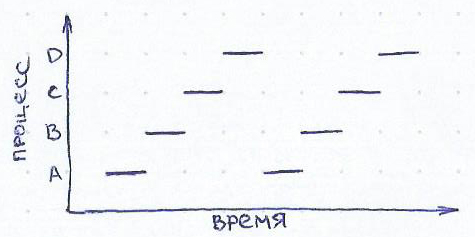


Рисунок 2 — Псевдопараллельная работа процессов

В Windows процессы являются контейнерами для программ. Они содержат виртуальное адресное пространство (виртуальную память), описатели объектов режима ядра, а также потоки. **Поток** — набор инструкций, выделенный в абстрактный блок. Как контейнеры для потоков они содержат также общие ресурсы, используемые для выполнения потоков, включая приоритет и другие различные маркеры для ядра. Каждый процесс имеет системные данные пользовательского режима, называемые PEB (Process Environment Block — блок среды процесса).

Процессы создаются из объектов сегментов, каждый из которых описывает объект памяти, основанный на дисковом файле. В Windows все процессы содержатся в строгой иерархии. Поэтому при создании нового процесса родительский процесс получает так называемый описатель, который позволяет ему контролировать дочерний процесс. Windows может объединять процессы в задания, группирующие процессы, чтобы ограничить содержащиеся в них потоки. Например, это можно использовать для более производительного обращения к общим ресурсам. Современная Windows объединяет не процессы, а потоки, ибо с первыми существуют значительные трудности организационного характера. Стоит отметить, что создание  
потока — критически ресурсоёмкая задача. Чтобы сделать работу операционной системы более быстрой, каждому процессу выделяется определённое количество потоков (пул, хранилище потоков), которые берут из очереди задания и выполняют их друг за другом, обеспечивая псевдо параллельность. Однако пул потоков имеет существенный недостаток: если текущая задача заблокирует дальнейший ход выполнения программ (например, программа ждёт, когда получит нужный ресурс), то он не сможет переключиться на другую задачу. Это происходит из-за того, что подобные ситуации блокируют не только задачу, но и работу пула в целом. Поэтому, пул будет создавать заведомо больше потоков, чтобы спланировать дальнейшее выполнение задач.

При запуске нового процесса Windows автоматически создаёт один единственный поток — основной поток программы. По мере необходимости процесс может "попросить" систему выделить некоторое количество потоков. Для этих целей программист в коде программы вызывает функцию создания потока ***CreateProcess***, предоставляемую самой Windows (для взаимодействия с операционной системой необходимо использовать Windows API, более известный как Win32). Важно отметить, что программист не создаёт поток с нуля вызовом одной лишь функции. Правильнее говорить, что мы запрашиваем систему создать поток и вернуть его сущность в качестве результата вызова функции. Алгоритм создания процесса можно описать следующей последовательностью действий:

1. Путь к исполняемому файлу (Executable, \*.exe) преобразуется в специальный NT формат, понятный ядру.
2. Подготавливаются необходимые параметры для запуска процесса.
3. Диспетчер процессов выделяет и инициализирует объект процесса (структуру данных, содержащую все необходимые данные).
4. Диспетчер памяти выделяет и подготавливает участок памяти в RAM, в рамках которого будет работать процесс.
5. Объект процесса добавляется в глобальный список процессов.
6. Объект процесса возвращается программисту в код.

Стоит заметить, что псевдо параллельность создаёт массу проблем, когда речь идёт об использовании какого-либо общего объекта данных. Часто возникает ситуация, которую можно объяснить на простом примере. Пусть в программе существует несколько потоков: один скачивает изображение из сети Интернет, а второй обрабатывает полученное изображение. Такая архитектура почти всегда будет оказываться неработоспособной. Это связано с тем, что потоки используют один и тот же ресурс в неопределённом порядке. На практике окажется, что поток, выполняющий обработку, будет пытаться выполнить свою работу с объектом изображения, которого ещё не существует. Дело в том, что этот поток ничего не знает о существовании второго и, следовательно, всегда ожидает, что объект готов к работе. В данном случае объект изображения не может быть готов исходя из соображений, что процессор работает быстрее, чем сетевая передача данных. Чтобы разрешить эту проблему, необходимо *синхронизировать* потоки.

Синхронизация потоков в Windows обеспечивается различными механизмами. В данной работе будет кратко изложена работа механизма под названием *семафор*. Семафор — это полноценный объект ядра, который создаётся при вызове функции ***CreateSemaphore***. Семафор представляет собой простой счётчик. Поток находится в заблокированном состоянии, пока значение его семафора отлично от нуля. Различные потоки увеличивают значение счётчика, сообщая, таким образом, о необходимости «подождать» завершения выполнения какой-либо задачи. Когда задача завершена, значение семафора уменьшается на единицу. Важно отметить, что если существуют четыре потока, три из которых в различное время увеличили счётчик оставшегося потока, то последний не продолжит свою работу, пока не завершится последняя задача и семафор не обнулится. В примере, который был приведён ранее, поток, осуществляющий загрузку изображения из сети, блокирует поток, который это изображение обрабатывает, путём увеличения значения семафора последнего на единицу. Когда загрузка будет завершена, то семафор потока-обработчика уменьшится (в данном случае — станет нулём) и запустится обработка изображения.

Ниже представлена обобщающая таблица 1 основных концепций, используемых для управления процессором и ресурсами.

Таблица 1 — Основные концепции, используемые для управления процессором и ресурсами

|  |  |
| --- | --- |
| **Название** | **Описание** |
| Задание | Коллекция процессов |
| Процесс | Контейнер для ресурсов |
| Пул потоков | Модель программирования, ориентированная на применение задач |

Со времён создания ЭВМ фон Неймана основная память компьютера организована как линейное адресное пространство, представляющее собой последовательность байт. Память — важнейший ресурс вычислительной системы. Важно подметить, ежегодное увеличение среднего объёма RAM не освобождает операционную систему от необходимости использовать память разумно. Ещё одним ключевым моментом является тот факт, что объём ОЗУ прямым образом влияет на производительность компьютера. Это связано с тем, что не все процессы работают параллельно в единый момент времени. Очевидно, что с высокой долей вероятности в данный момент существует как минимум один процесс, ожидающий, например, завершения операции ввода/вывода. Довольно неразумно оставлять приостановленный процесс в памяти. В таком случае операционная система сохраняет образ данного процесса на диск и в освободившийся участок памяти подгружает другой образ (процесс), который готов к работе.

Чтобы абстрагировать концепцию памяти и не усложнять разработку кода, существует идея **виртуальной памяти**. Виртуальная память даёт возможность программисту думать, что оперативная память бесконечна. Задача операционной системы — реализовать концепцию бесконечной памяти в рамках физического объёма RAM. Чтобы решить данную задачу, были разработаны многочисленные идеи распределения памяти. В ОС Windows реализована **страничная организация памяти**, концептуальная работа которой изложена ниже.

Страничная организация памяти является вполне удачным и часто используемым подходом к организации ОЗУ. Чтобы внести ясность в дальнейшее объяснение материала, необходимо понимать, что **виртуальное адресное пространство** — адреса, к которым может обращаться программа (до 4 Гбайт на 32-х битной архитектуре), а **физическое адресное пространство** — реальные адреса памяти, реализованные аппаратно. Виртуальное адресное пространство разделяется на ряд страниц (так называемые **виртуальные страницы**); физическое адресное пространство также разбивается на части равного размера таким образом, чтобы каждая такая часть основной памяти вмещала ровно одну страницу. Эти части основной памяти называются **страничными кадрами**. Каждый компьютер с виртуальной памятью содержит устройство для отображения виртуальных адресов на физические — диспетчер памяти (Memory Management Unit, MMU). Когда виртуальный адрес необходимо преобразовать в физический адрес, то вначале просматривается **буфер быстрого преобразования адресов** (Translation Lookaside Buffer, TLB). Если совпадение найдено, то возвращается физический адрес, и доступ к памяти может продолжаться. Однако если нет совпадения, то либо блок управления памятью либо обработчик пропусков TLB операционной системы ищет сопоставление адресов в таблице страниц (элемент таблицы страниц проиллюстрирован на рисунке 3), чтобы узнать, существует ли сопоставление  
(за это отвечает **бит присутствия**). Если сопоставление существует — оно записывается обратно в TLB (это необходимо сделать, поскольку аппаратное обеспечение обращается к памяти через TLB в системе виртуальной памяти), и текущая команда преобразования адреса перезапускается (что также может происходить параллельно). На рисунке 4 проиллюстрирован процесс преобразования адресов на аппаратном уровне.



Рисунок 3 — Элемент таблицы страниц (page table entry, PTE)  
отображенной страницы для архитектур  
Intel x86 и AMD x64

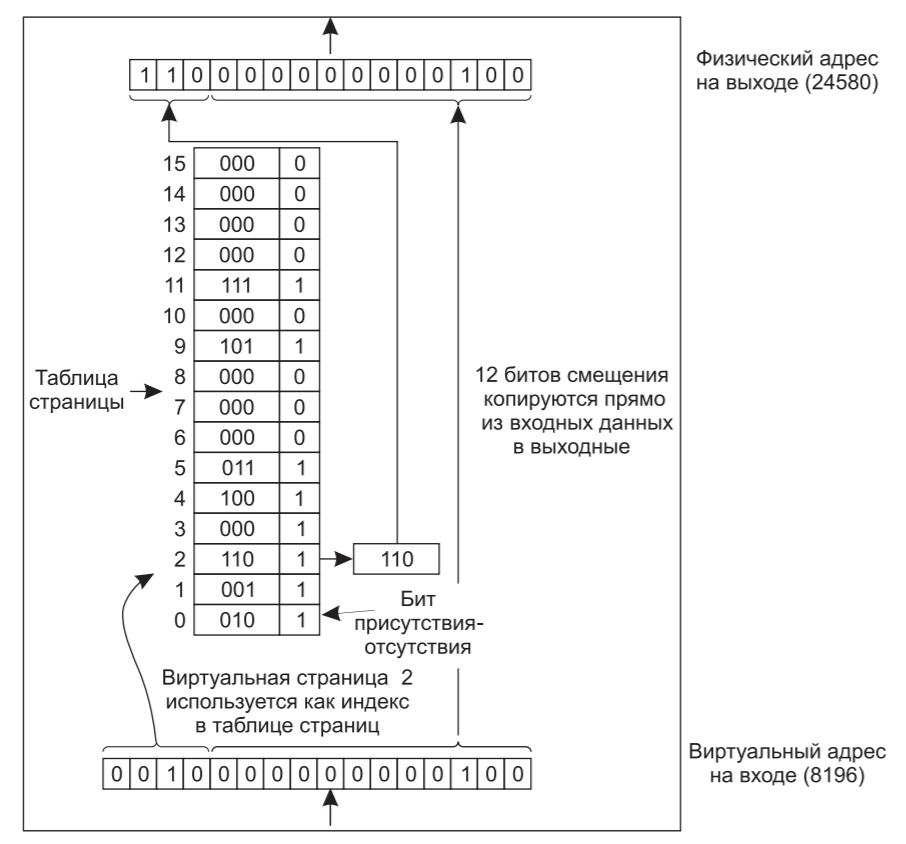


Рисунок 4 — Преобразование диспетчером памяти  
виртуального адреса в физический адрес

Общая схема организации памяти в ОС Windows представлена на рисунке ниже.

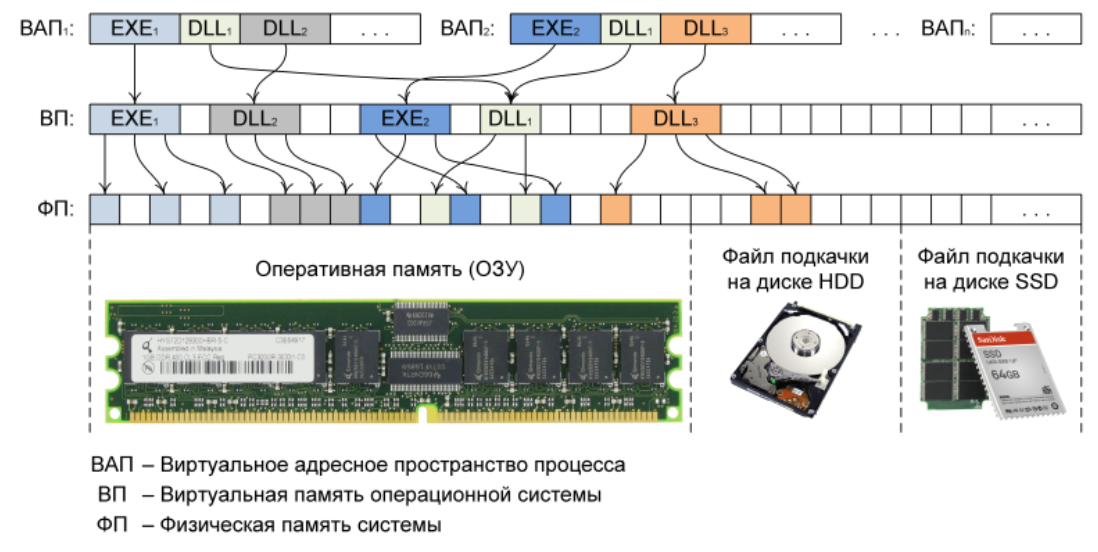


Рисунок 5 — Схема организации памяти в ОС семейства Windows NT

Обмен данными между пользователями, приложениями и периферийными устройствами специальная ОС – подсистема ввода-вывода. Основными компонентами подсистемы ввода-вывода являются **драйверы**, управляющие внешними устройствами, и **файловая система**. В работе подсистемы ввода-вывода активно участвует диспетчер прерываний. Более того, основная нагрузка диспетчера прерываний обусловлена именно подсистемой ввода-вывода, поэтому диспетчер прерываний иногда считают частью подсистемы ввода-вывода.

Файловая система – это основное хранилище информации в любом компьютере. Она активно использует остальные части подсистемы ввода-вывода. Кроме того, модель файла лежит в основе большинства механизмов доступа к периферийным устройствам.

Подсистема ввода-вывода выполняет следующую функциональность:

1. грамотная организация параллельной работы устройств ввода-вывода и процессора;
2. согласование скоростей обмена и кэширования данных;
3. разделение устройств и данных между процессами;
4. поддержка драйверов и их динамическая загрузка;
5. поддержка нескольких файловых систем;
6. поддержка синхронных и асинхронных операций ввода-вывода.

По мере исторического развития и совершенствования архитектуры компьютера центральный процессор все меньше и меньше вмешивается в функции ввода-вывода. Благодаря этому повышается общая производительность системы.

Для персональных компьютеров операции ввода-вывода могут выполняться тремя способами.

1. С помощью программируемого ввода-вывода. В этом случае, когда процессору встречается команда, связанная с вводом-выводом, он выполняет её, посылая соответствующие команды контроллеру ввода-вывода. Это устройство выполняет требуемое действие, а затем устанавливает соответствующее состояние ввода-вывода и не посылает никаких сигналов, в том числе сигналов прерываний (прерывание – команда процессору немедленно приостановить выполнение текущей задачи и выполнить инструкции прерывания, которые содержатся в специальной таблице прерываний под определённым номером). Процессор периодически проверяет состояние модуля ввода-вывода с целью проверки завершения операции ввода-вывода.

Таким образом, процессор непосредственно управляет операциями ввода-вывода, включая опознание состояния устройства, пересылку команд чтения-записи и передачу данных. Процессор посылает необходимые команды контроллеру ввода-вывода и переводит текущий процесс в состояние ожидания завершения операции ввода-вывода. Недостатки такого метода – большие потери процессорного времени, связанные с управлением вводом-выводом.

1. Ввод-вывод, управляемый прерываниями. Процессор посылает необходимые команды контроллеру ввода-вывода и продолжает выполнять текущий процесс, если нет необходимости в ожидании завершения выполнения текущей операции ввода-вывода. В противном случае текущий процесс приостанавливается до получения сигнала прерывания о завершении ввода-вывода, а процессор переключается на выполнение другого процесса. Наличие прерываний процессор проверяет в конце каждого цикла выполняемых команд.

Такой ввод-вывод намного эффективнее, чем программируемый ввод-вывод, так как при этом исключается ненужное ожидание с бесполезным простоем процессора. Однако и в этом случае ввод-вывод потребляет ещё значительное количество процессорного времени, потому что каждое слово (единица данных), которое передаётся из памяти в модуль ввода-вывода (контроллер) или обратно, должно пройти через процессор.

1. Прямой доступ к памяти (direct memory access – DMA). В этом случае специальный модуль прямого доступа к памяти управляет обменом данных между основной памятью и контроллером ввода-вывода. Процессор посылает запрос на передачу блока данных модулю прямого доступа к памяти, а прерывание происходит только после передачи всего блока данных.

В настоящее время в персональных и других компьютерах используется третий способ ввода-вывода, поскольку в структуре компьютера имеется DMA-контроллер или подобное ему устройство, обслуживающее, как правило, запросы по передаче данных от нескольких устройств ввода-вывода на конкурентной основе. Процесс работы DMA-контроллера представлен на рисунке ниже.

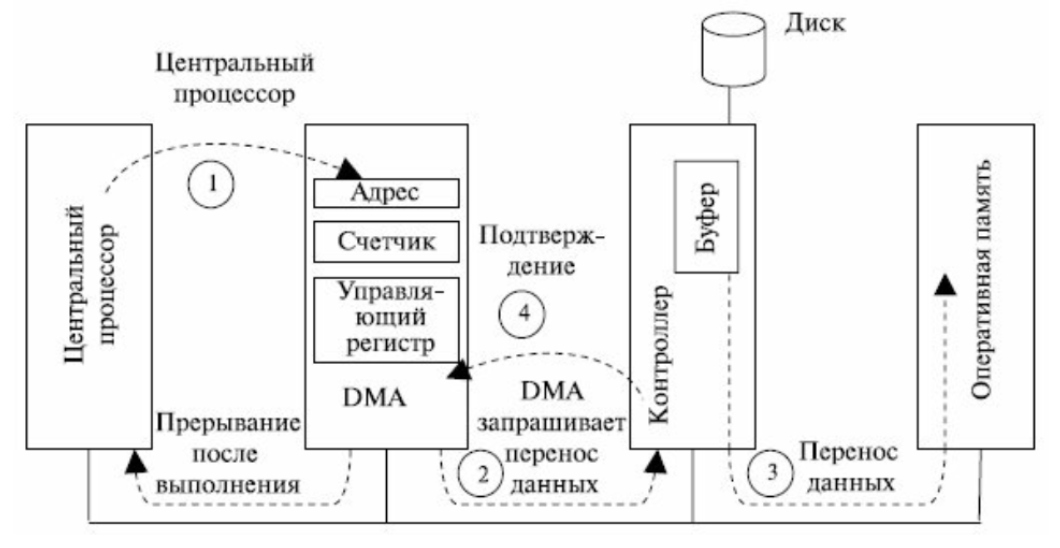


Рисунок 6 — Работа DMA

Первым шагом процессор программирует контроллер (шаг 1), чтобы сделать его работу корректным для предстоящей операции, а затем даёт команду дисковому контроллеру и внести данные во внутренний буфер. После этого продолжается работа процессора, а следом – контроллера DMA. DMA начинает перенос данных (шаг 2), записывая их в буфер дискового контроллера. Наконец данные из буфера будут перенесены в ОЗУ контроллером (шаг 3). Когда запись в ОЗУ завершена, контроллер памяти посылает DMA сигнал подтверждения.

Исходя из вышеизложенного материала, следует отметить, что Microsoft Windows представляет собой сложную операционную систему, состоящую из огромного количества взаимосвязанных компонентов, обеспечивающих стабильную работу как аппаратного, так и прикладного уровня ЭВМ в большинстве случаев. Учитывая тенденции аппаратных требований современных программных решений, важно грамотно пользоваться ресурсами системы. Вышеизложенный материал является минимум фундаментальных знаний о внутреннем устройстве операционных систем как таковых. Несмотря на то, что многие операционные системы существуют не одно десятилетие, программисту следует использовать ресурсы компьютера максимально рационально, руководствуясь знаниями внутреннего устройства той системы, под которую ведётся разработка программного продукта.

# ГЛАВА 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

Программный продукт был разработан с использованием языка программирования C++/CLI в среде Microsoft Visual Studio. Чтобы запустить программу, необходимо установить её на локальный компьютер. Для этого следует воспользоваться установщиком setup.exe (убедитесь, что также присутствует файл Installer.msi).

## 2.1 Разработка программы (разработка электронного конспекта)

Для того, чтобы разработать электронный конспект, потребовалось создать html-документ, в котором был отражён теоретический материал по вышеупомянутой теме.

На главной форме приложения (рисунок 7) представлена кнопка «НАЧАТЬ РАБОТУ», которая вызывает окно авторизации для осуществления ввода пользовательских данных. Это показано на рисунке 8.

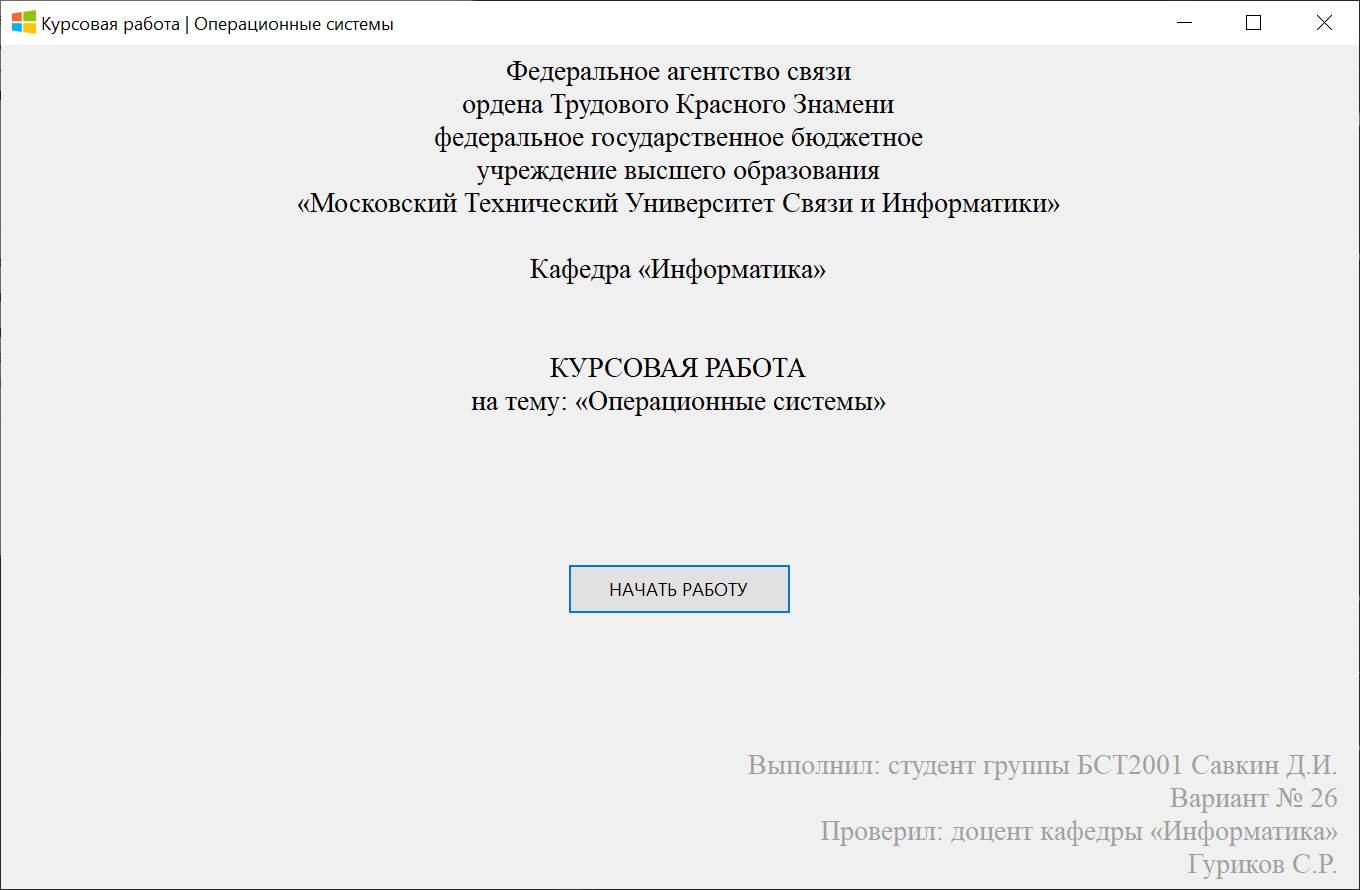


Рисунок 7 — Главное окно программы

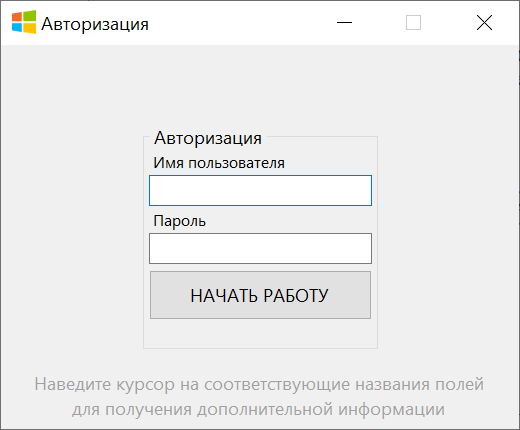


Рисунок 8 — Окно авторизации

Пройдя авторизацию, пользователь попадёт на страницу, где может ознакомиться с теоретическим материалом (рисунок 9). При переходе на форму, пользователь получит уведомление в следующем виде:

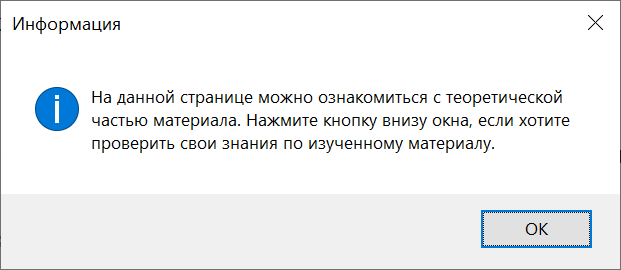


Рисунок 9 — Уведомление при переходе на окно с теоретическим материалом

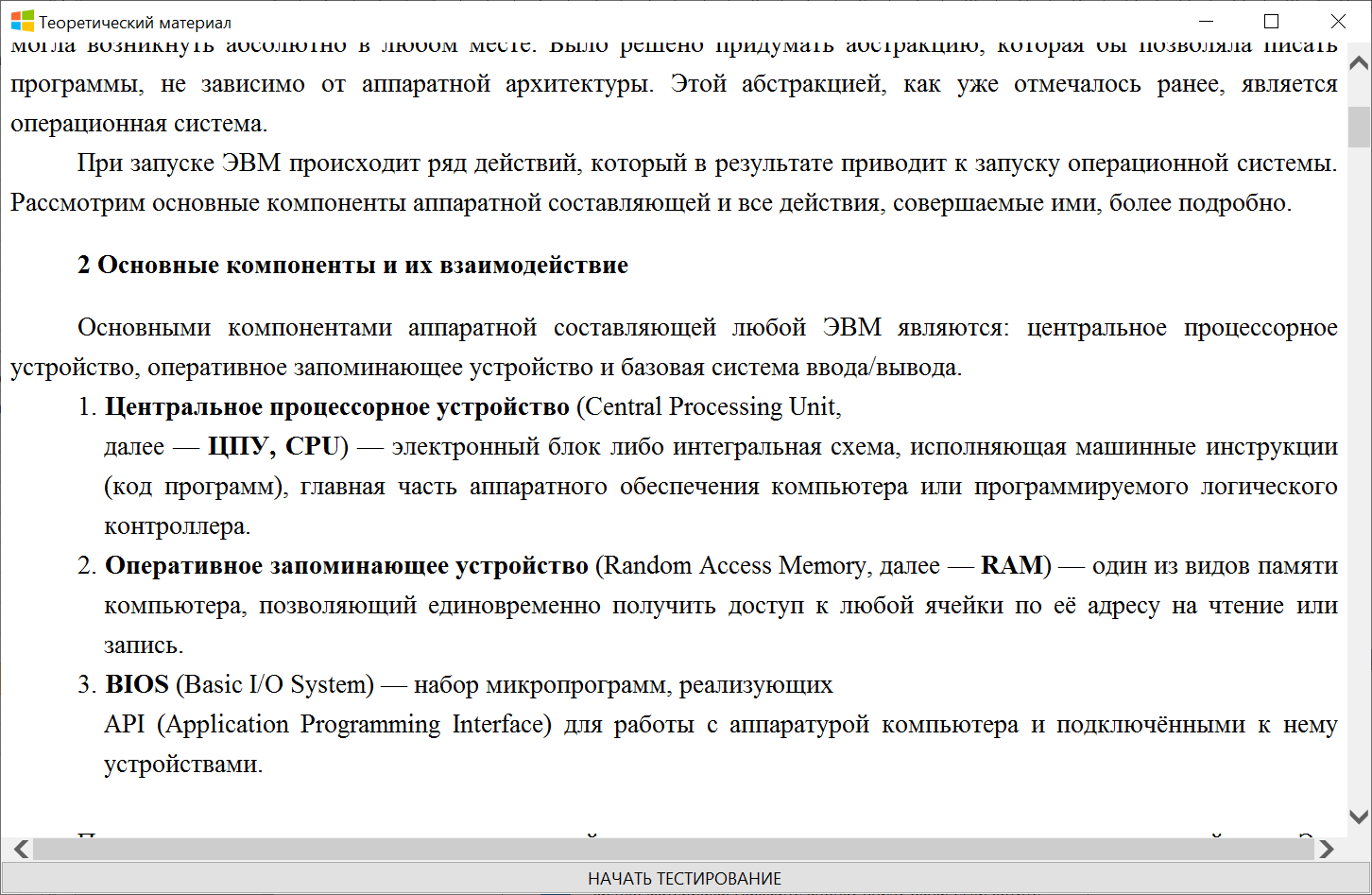


Рисунок 10 — Окно с теоретическим материалом

Материал выводится с помощью элемента WebBrowser, динамически загружая информацию из html-документа. С помощью кнопки «НАЧАТЬ ТЕСТИРОВАНИЕ» пользователь может перейти к блоку тестирования.

## 2.2 Разработка программы (разработка тестовой части)

Пользователь может беспрепятственно переходить к следующему вопросу, не отвечая на текущий. Система автоматически проверяет выбранный по умолчанию ответ на правильность. Если же ответ по умолчанию не задан, то считается, что пользователь ответил неправильно. Чтобы начать тестирование, пользователю необходимо снова подтвердить данные пользователя, введя их в появившемся окне авторизации, которое было показано ранее. Блок тестирования состоит из 16 вопросов, на которые должен ответить пользователь. Вопросы условно дифференцируются на категории, разделённые по принципу ввода пользовательского ответа. В большинстве случаев необходимо выбрать верный ответ из предложенных вариантов, представляющих собой блок из элементов RadioButton, ListBox, или несколько ответов из предложенных в списке (CheckedListBox и блок элементов CheckBox), установив соответствующее количество флажков. Указанные способы ввода ответа показаны на рисунках 11, 12, 13 и 14 соответственно.

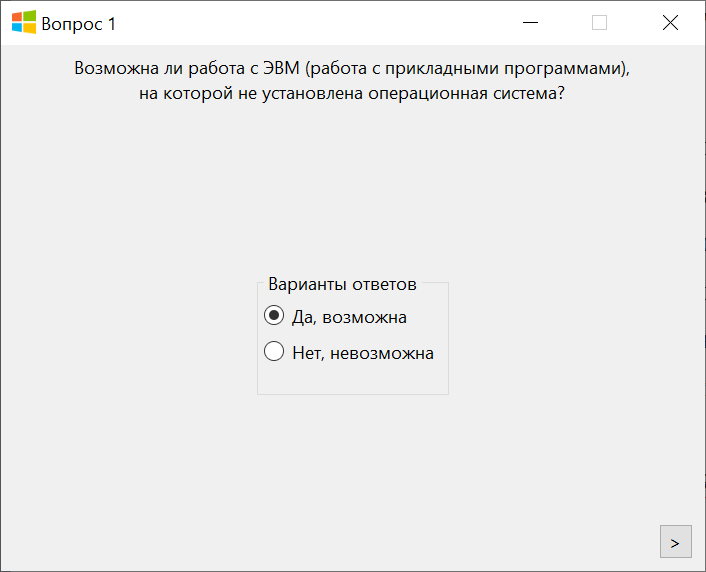


Рисунок 11 — Выбор ответа из предложенных  
вариантов (RadioButton)

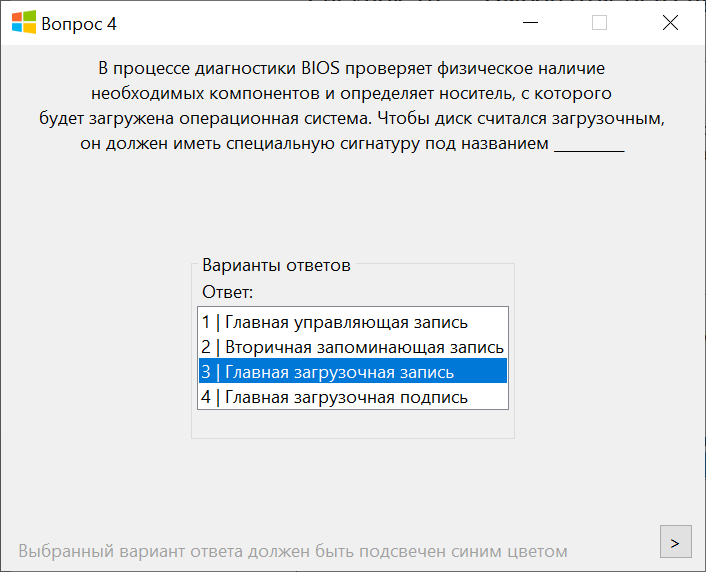


Рисунок 12 — Выбор ответа из списка (ListBox)

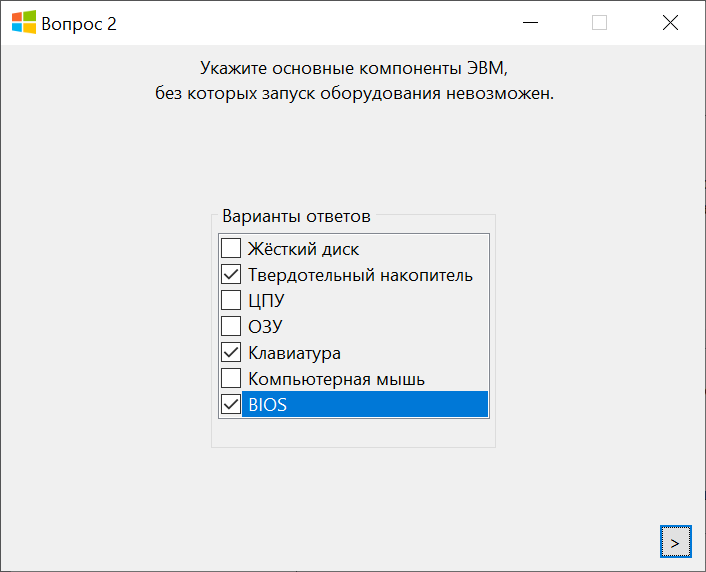


Рисунок 13 — Выбор нескольких ответов (CheckedListBox)

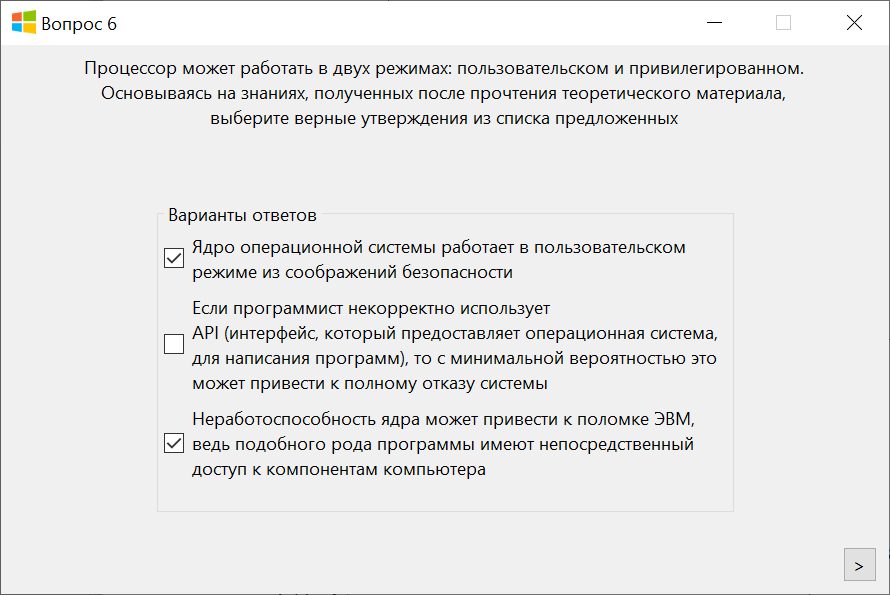


Рисунок 14 — Выбор нескольких ответов (CheckBox)

Некоторые вопросы требуют ввода ответа в текстовом виде (рисунок 15). Стоит заметить, что программа способна отформатировать введённый текст в поле ввода TextBox. Таким образом, пользователь может вводить ответ в любом регистре, а пробелы до и после текста игнорируются.

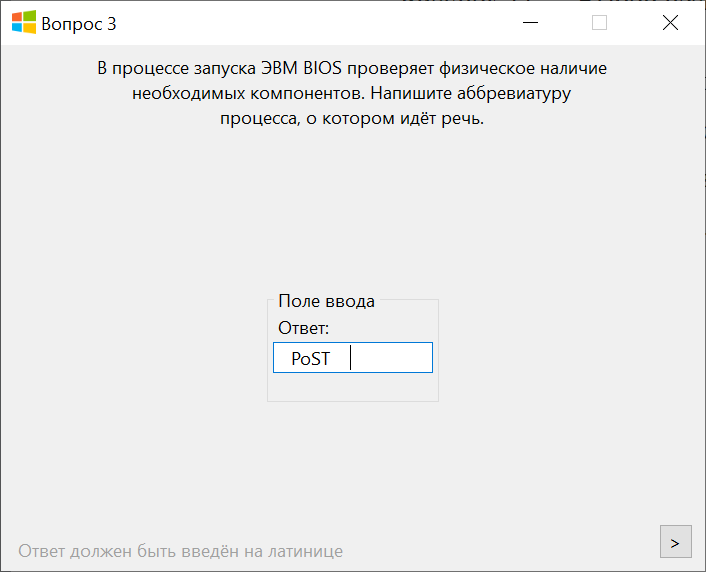


Рисунок 15 — Ввод ответа с клавиатуры

Вопрос №5 требует от пользователя перемещения бегунока полосы прокрутки элемента HScrollBar «под» нужный вариант ответа. Чтобы сделать ответ более простым и понятным, программа выделяет полужирным шрифтом выбранный элемент. Это показано на рисунке ниже.

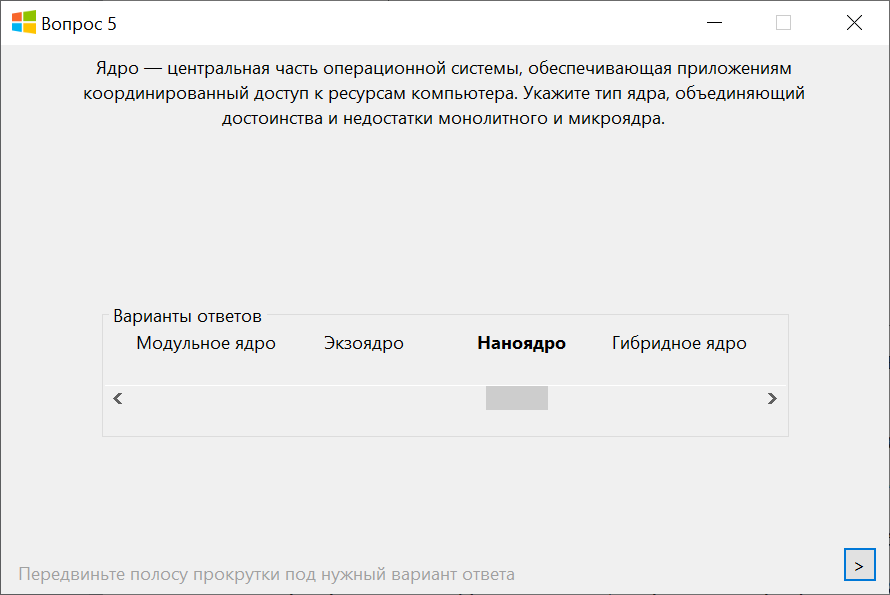


Рисунок 16 — Полоса прокрутки в роли указателя ответа

Вопрос №7 предлагает пользователю указать нужное число в указанных границах. Подобно взаимодействию с полосой прокрутки, пользователю следует переместить указатель элемента TrackBox на нужный вариант ответа. Выбранный ответ выделяется полужирным шрифтом:

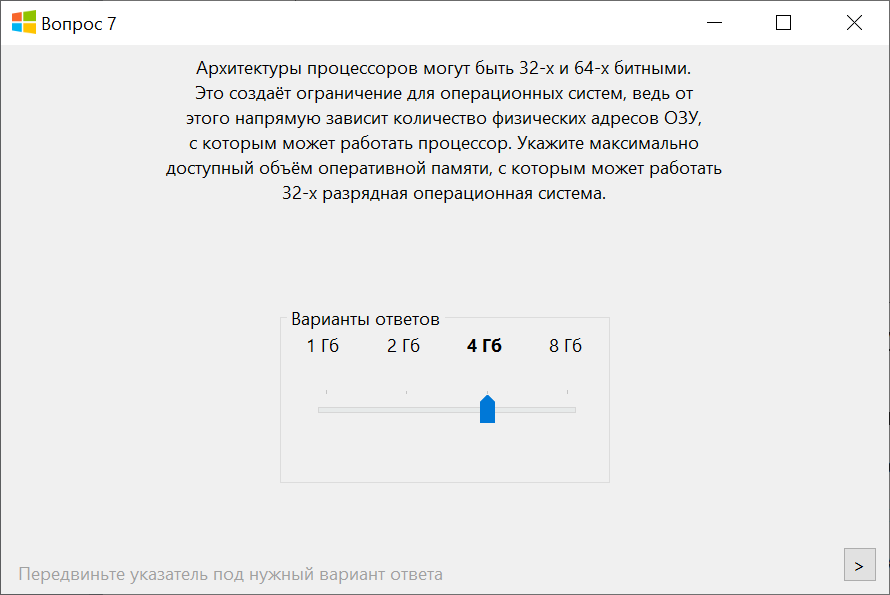


Рисунок 17 — Слайдер в качестве ввода ответа пользователя

Вопрос №12 требует от пользователя формирования связи между названием концепции и её описанием. Для этого пользователь указывает соответствующий номер описания рядом с названием в элементе NumericUpDown. Минимальное и максимальное число, которое может быть задано в элементе — это 1 и 3 соответственно. Это показано на рисунке ниже.

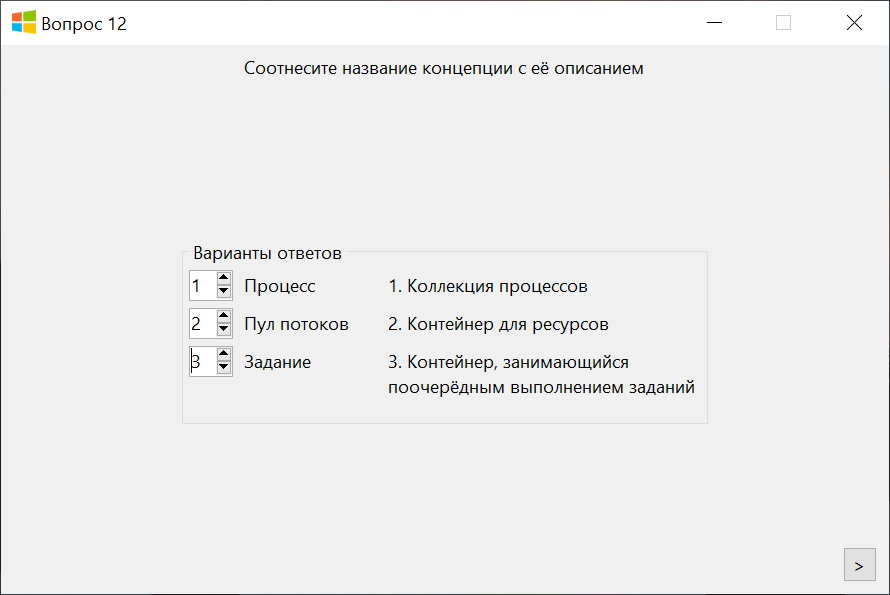


Рисунок 18 — Использование счётчика в качестве  
связующей структуры

В программе реализовано уведомление о правильности ответа. Если ответ верный, то на экране появляется следующее окно:

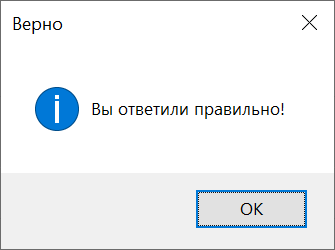


Рисунок 19 — Уведомление о верном ответе

В случае, когда пользователь отвечает неверно, выводится окно с пояснением правильного ответа:

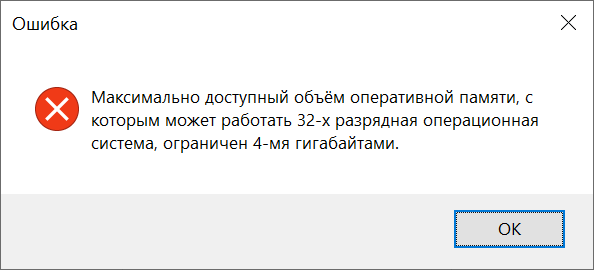


Рисунок 20 — Уведомление о неверном ответе (вопрос №7)

После прохождения всех вопросов пользователь попадает в окно с результатами тестирования. Цвет окна визуально интерпретирует полученный результат в зависимости от процента выполнения: розовым от 0 до 33, жёлтым от 33 до 66 и зелёным от 66 до 100. Это показано на рисунках 21, 22 и 23 соответственно.

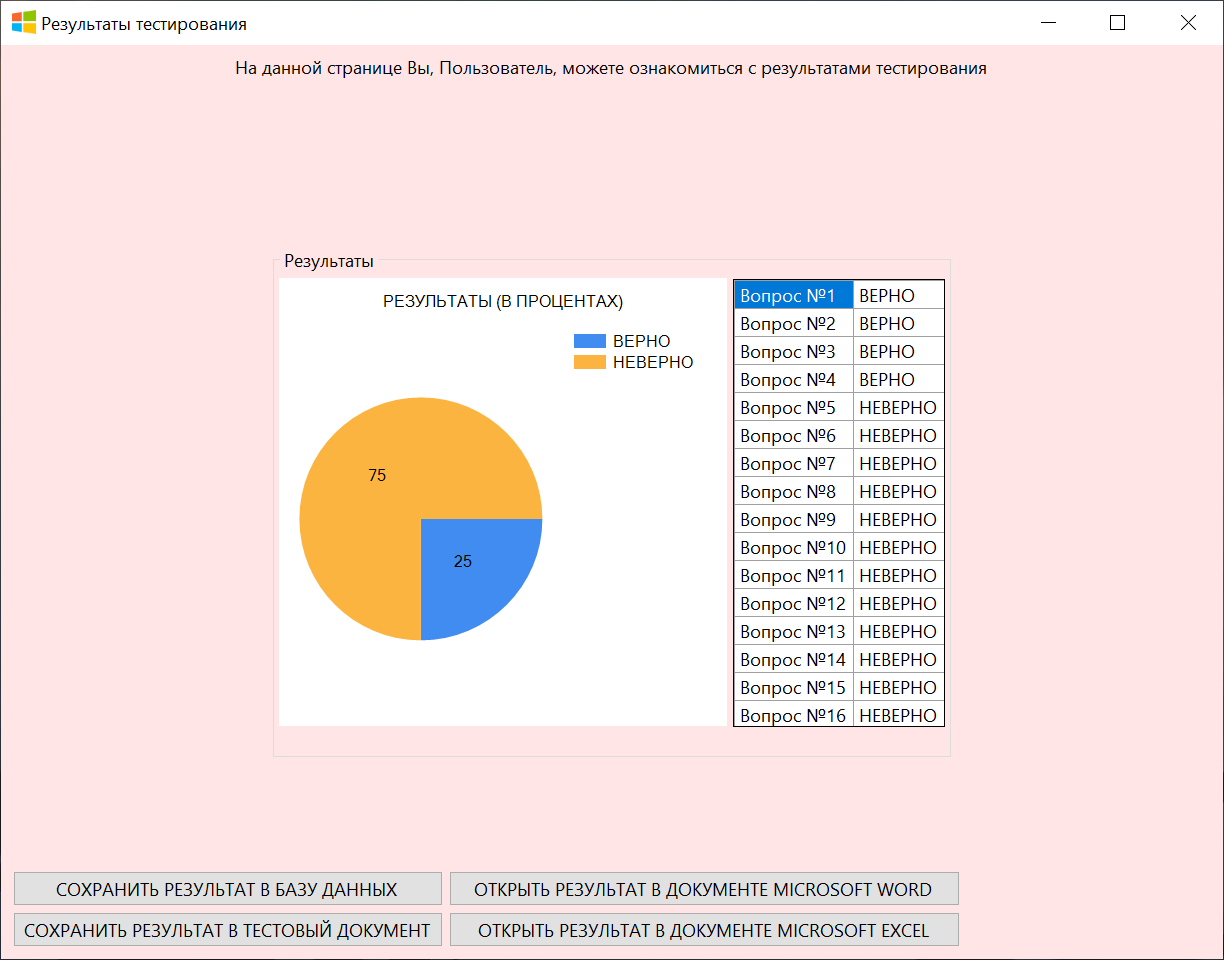


Рисунок 21 — Низкий процент выполнения

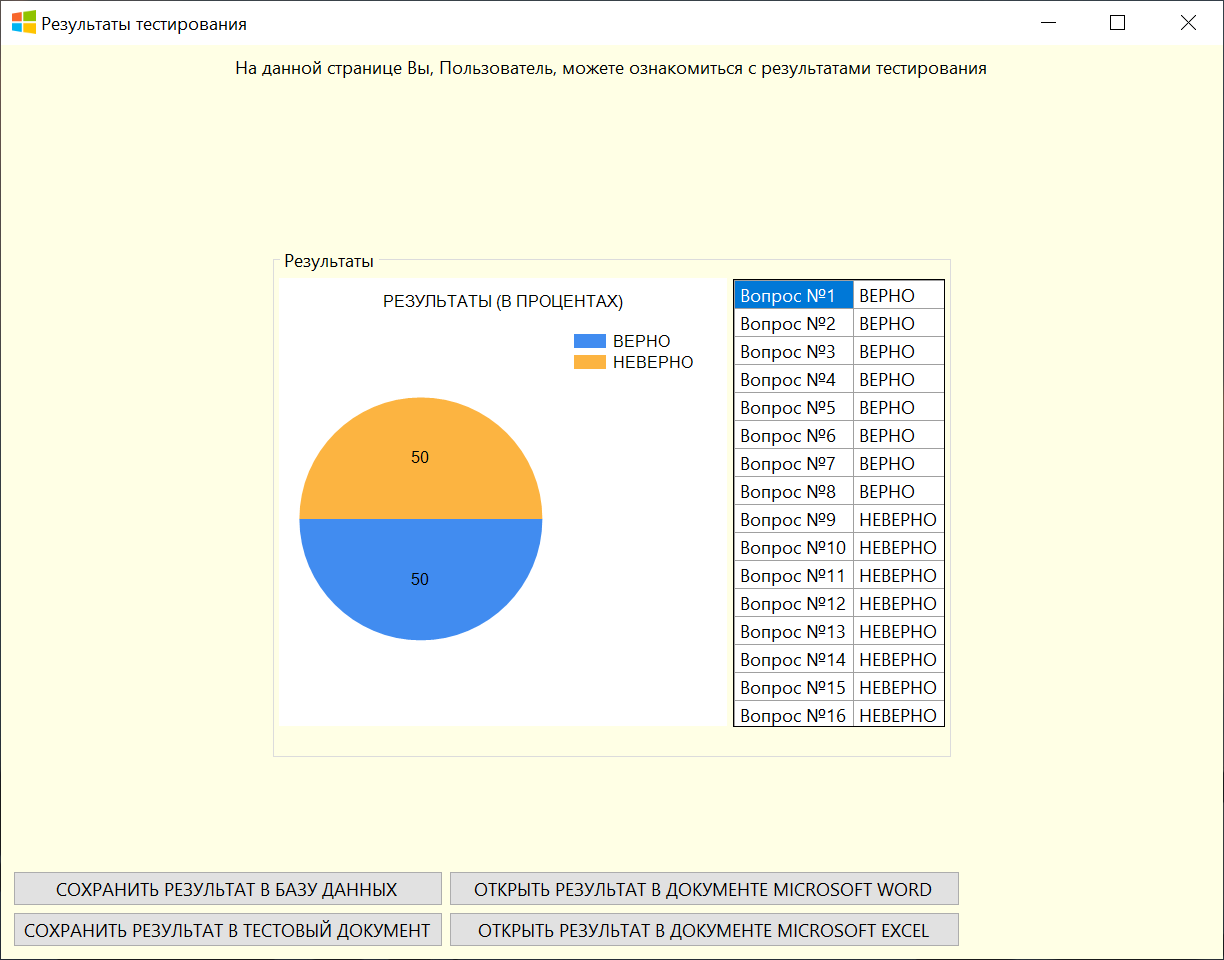


Рисунок 22 — Средний процент выполнения

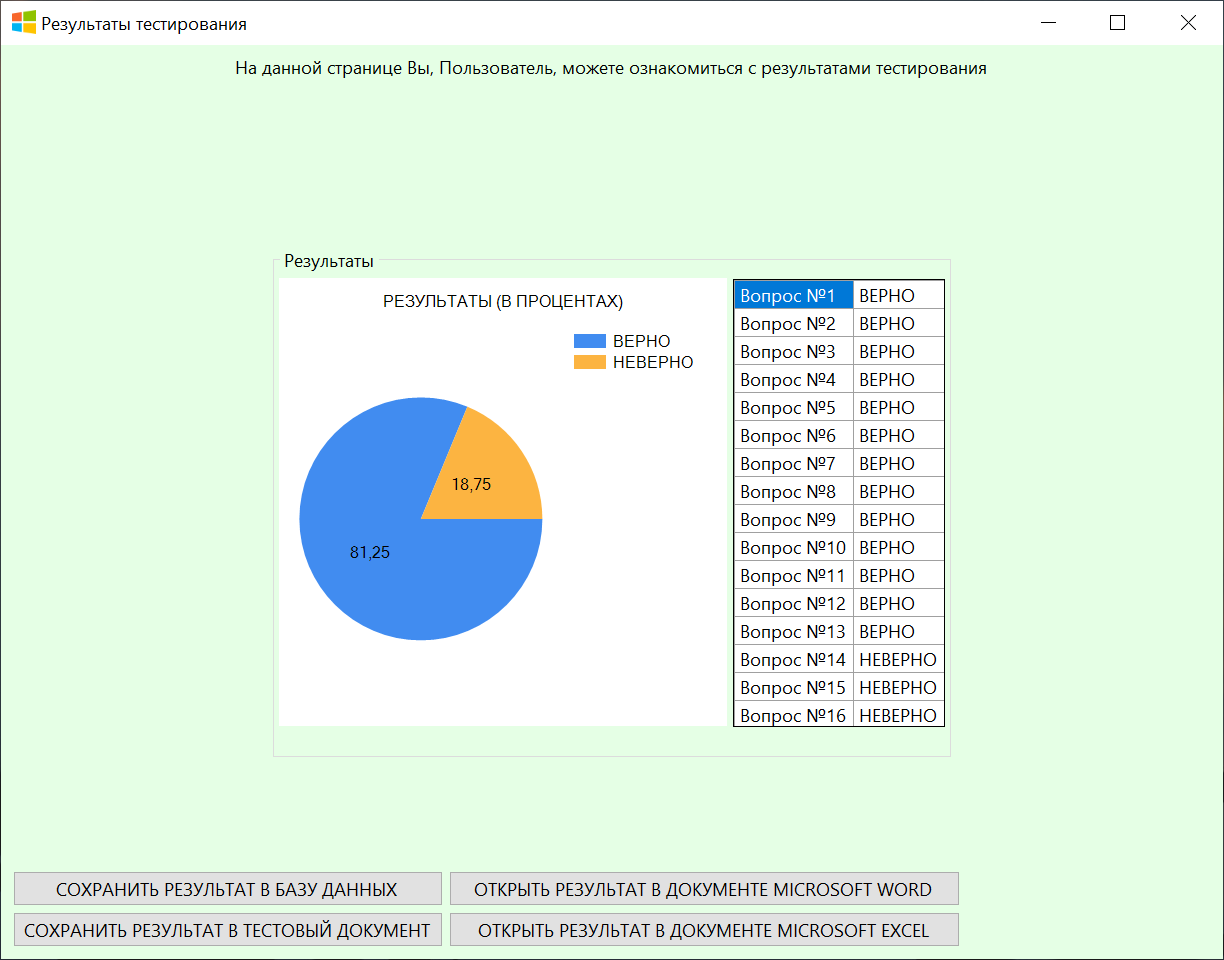


Рисунок 23 — Высокий процент выполнения

На экране результатов тестирования пользователь видит визуальное отображение соотношение верных и неверных ответов в элементе Chart. Более конкретная сводка представлена в виде таблицы в элементе DataGridView. Также пользователь может сохранить результаты тестирования в любом удобном для него формате. Для этого используются одноимённые кнопки внизу экрана.

Код, реализующий вопросы и соответствующие интерфейсы взаимодействия, представлен в приложении Б.

## 2.3 Дополнительные возможности программы

Разработка программного продукта курсовой работы предполагала реализацию следующих дополнительных возможностей с целью грамотной организации архитектуры приложения и обеспечения удобства его разработки.

В частности, было реализовано:

1. оформление кода всех вопросов в виде функций, возвращающих результат;
2. создание DLL-библиотеки, содержащую код всех вопросов из предыдущего пункта;
3. разработка системы авторизации;
4. организация вывода результатов тестирования в одномерный массив с выводом в DataGridView;
5. организация вывода результатов тестирования в базу Microsoft Access;
6. организация вывода результатов в тестовый документ;
7. организация вывода результатов тестирования в документ Microsoft Word;
8. организация вывода результатов тестирования в лист Microsoft Excel;
9. организация краткого вывода результатов в виде диаграммы;
10. графическая заставка.

Рассмотрим вышеперечисленные возможности более подробно.

Система авторизации представляет собой окно, в котором пользователю необходимо ввести своё имя и пароль. Имя пользователя используется для вывода результата — вверху соответствующего окна выводится текст, содержащий обращение к пользователю. На рисунках 21, 22 и 23 можно увидеть обращение к пользователю, если его имя «Пользователь» соответственно. Поле ввода имени пользователя не может быть пустым (в противном случае система выдаст соответствующую ошибку).

Программный код, реализующий данную возможность:

Код библиотеки, проверяющий правильность ввода данных:

bool CourseworkLibrary::Tools::isCredentialsValid(TextBox^ field\_username, TextBox^ field\_password)

{

String^ username = field\_username->Text->Trim();

String^ password = field\_password->Text->Trim();

if (username->Equals(""))

{

showErrorBox("Поле имени пользователя не может быть пустым");

return false;

}

if (!password->Equals("password1324"))

{

showErrorBox("Неверный пароль");

return false;

}

return true;

}

Класс, управляющий авторизацией пользователя:

// User.h

#pragma once

public ref class User

{

private:

static User instance;

System::String^ username = "Пользователь";

bool state = false;

private:

User() {}

User(const User%);

public:

static property User^ Instance

{

User^ get()

{

return %instance;

}

}

void logIn(System::String^ username);

System::String^ getName();

bool isLoggedIn();

bool checkAuth(System::String^ username);

};

// User.cpp

#include "User.h"

User::User(const User%)

{

// Объект пользователя в куче должен быть один

throw gcnew System::InvalidOperationException("Cannot have more than one instances of class");

}

void User::logIn(System::String^ username)

{

this->username = username;

this->state = true;

}

System::String^ User::getName()

{

return username;

}

bool User::isLoggedIn()

{

return state;

}

bool User::checkAuth(System::String^ username)

{

return !this->state || this->username->Equals(username);

}

В указанном коде используется шаблон проектирования «Одиночка» («Singleton»), что позволяет сделать экземпляр класс глобальным.

// Auth.h form

// Окно авторизации проверяет корректность данных

// и выполняет переход на следующую форму

System::Void button\_start\_click(System::Object^ sender, System::EventArgs^ e)

{

String^ username = field\_username->Text->Trim();

if (Tools::isCredentialsValid(field\_username, field\_password))

{

User^ user = User::Instance;

if (!user->checkAuth(username))

{

Tools::showErrorBox("Введенное имя пользователя не совпадает с именем, зарегистрированным " +

"ранее!\nПроверьте корректность ввода.");

return;

}

User::Instance->logIn(username);

Tools::showInfoBox("Аутентификация выполнена успешно!");

parent->Hide();

this->Hide();

next->Show();

}

}

Организация вывода результатов тестирования в одномерный массив в элемент DataGridView происходит с сопутствующим форматированием. Программный код, реализующий данную возможность:

Код библиотеки:

void CourseworkLibrary::Tools::showResults(DataGridView^ view, array<bool>^ results)

{

view->ColumnCount = 2;

view->RowCount = 16;

for (size\_t i = 0; i < 16; i++)

{

view->Rows[i]->Cells[0]->Value = "Вопрос №" + (i + 1);

view->Rows[i]->Cells[1]->Value = results[i] ? "ВЕРНО" : "НЕВЕРНО";

}

int sum = 0;

for (int i = 0; i < view->RowCount; i++)

sum += view->Rows[i]->Height;

view->Height = sum;

view->Columns[0]->AutoSizeMode = DataGridViewAutoSizeColumnMode::AllCells;

view->Columns[1]->AutoSizeMode = DataGridViewAutoSizeColumnMode::AllCells;

}

Код программы:

// Results.h

// Вызывается после загрузки интерфейса

System::Void onLoad(System::Object^ sender, System::EventArgs^ e)

{

label\_title->Text = "На данной странице Вы, " + User::Instance->getName()

", можете ознакомиться с результатами тестирования";

// Получаем процент выполнения

double percentage = UserAnswers::Instance->getPercentage();

// Вывод результатов в элементы

Tools::showResults(result\_dgv, answers);

Tools::showResults(result\_chart, percentage);

// Цвет фона соответствует проценту выполнения

Tools::setBackgroundColor(this, percentage);

}

Организация вывода результатов тестирования в базу Microsoft Access выполнена с соответствующим форматированием:

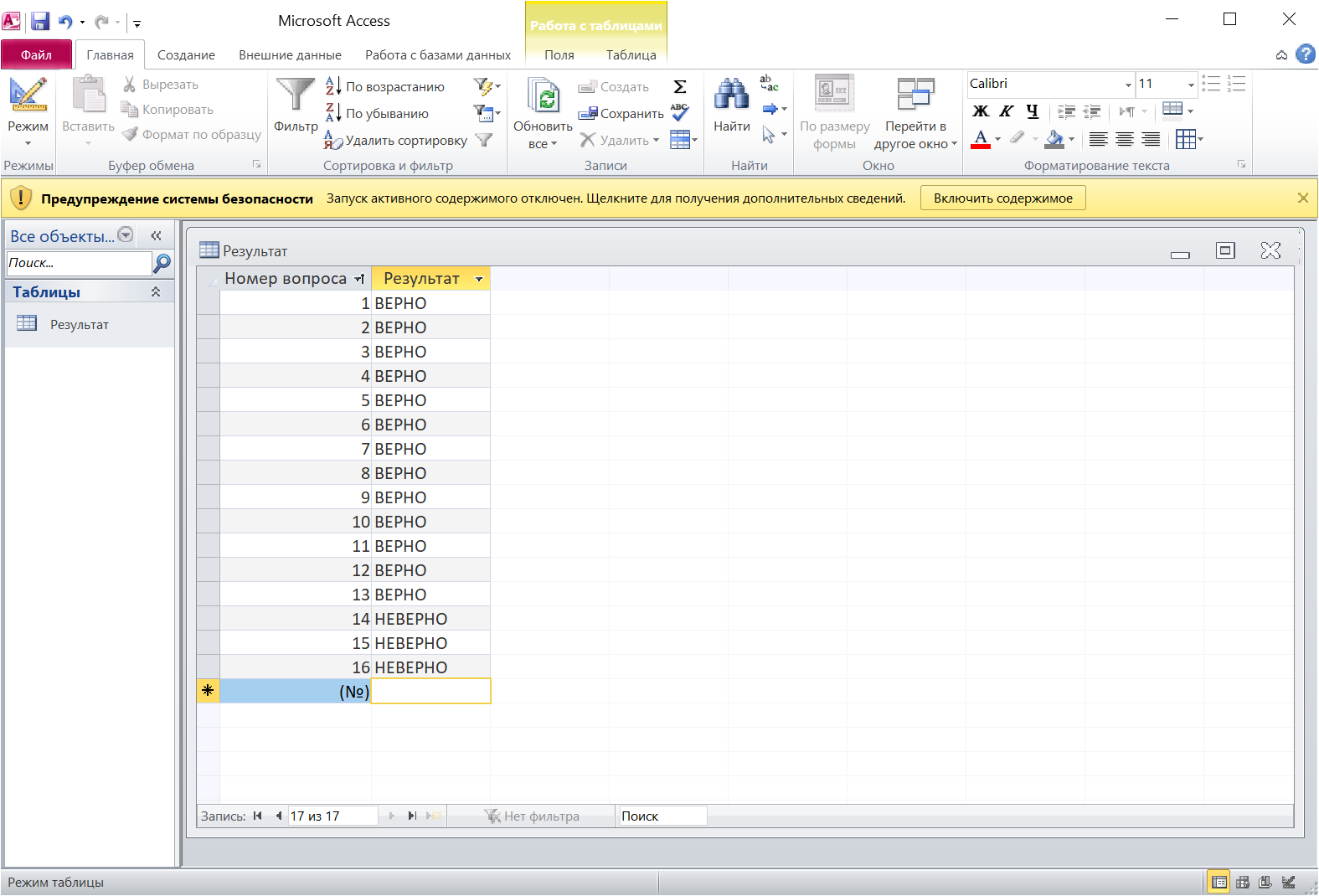


Рисунок 24 — Вывод результатов в базу Microsoft Access

Программный код, реализующий данную возможность:

Код библиотеки:

static void create\_db(String^ path)

{

ADOX::Catalog^ catalog = gcnew ADOX::CatalogClass();

try

{

catalog->Create("Provider = Microsoft.Jet.OLEDB.4.0; Data Source = " + path);

}

catch (System::Runtime::InteropServices::COMException^ exception)

{

MessageBox::Show(exception->Message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

finally

{

catalog = nullptr;

}

}

static void configure\_db(String^ path)

{

auto p = gcnew OleDbConnection("Provider = Microsoft.Jet.OLEDB.4.0; Data Source = " + path);

p->Open();

auto c = gcnew OleDbCommand("CREATE TABLE [Результат]([Номер вопроса] counter, [Результат] char(200))", p);

try

{

c->ExecuteNonQuery();

}

catch (OleDbException^ exception)

{

MessageBox::Show(exception->Message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

p->Close();

}

void CourseworkLibrary::Tools::write\_to\_db(array<bool>^ array, String^ timestamp)

{

String^ path = buildFolders(timestamp) + "\\РЕЗУЛЬТАТ.mdb";

create\_db(path);

configure\_db(path);

for (size\_t i = 0; i < 16; i++)

{

auto p = gcnew OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Data Source = " + path);

p->Open();

auto c = gcnew OleDbCommand("INSERT INTO[Результат](" +

"[Номер вопроса], [Результат]) VALUES('" + (i + 1) + "','" + (array[i] ? "ВЕРНО" : "НЕВЕРНО") + " ')");

c->Connection = p;

c->ExecuteNonQuery();

}

showInfoBox("Ваши результаты были успешно записаны в таблицу Microsoft Access. Файл отчета может быть найден" +

"в директории results рядом с запускаемым файлом программы.");

}

Код программы:

System::Void save\_as\_db(System::Object^ sender, System::EventArgs^ e)

{

String^ tmp = button\_save\_as\_db->Text;

button\_save\_as\_db->Text = L"СОХРАНЕНИЕ...";

this->Enabled = false;

Tools::write\_to\_db(answers, timestamp);

this->Enabled = true;

button\_save\_as\_db->Text = tmp;

}

Организация вывода результатов в тестовый документ происходит с сопутствующим форматированием (рисунок 25). Программный код, реализующий данную возможность:

Код библиотеки:

void CourseworkLibrary::Tools::write\_to\_notepad(array<bool>^ array, String^ timestamp)

{

String^ path = buildFolders(timestamp) + "\\РЕЗУЛЬТАТ.txt";

try {

auto encoding = System::Text::Encoding::GetEncoding(1251);

auto stream = gcnew StreamWriter(path, false, encoding);

stream->WriteLine("Номер вопроса | Результат");

for (int i = 0; i < 16; i++)

{

stream->WriteLine(String::Format("{0, 16}", (i + 1) + " | ") + (array[i] ? "ВЕРНО" : "НЕВЕРНО"));

}

stream->Close();

showInfoBox("Ваши результаты были успешно записаны в тестовый файл. Файл отчета может быть найден" +

"в директории results рядом с запускаемым файлом программы.");

}

catch (System::Exception^ exception) {

MessageBox::Show(exception->Message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

Код программы:

System::Void save\_as\_txt(System::Object^ sender, System::EventArgs^ e)

{

Tools::write\_to\_notepad(answers, timestamp);

}

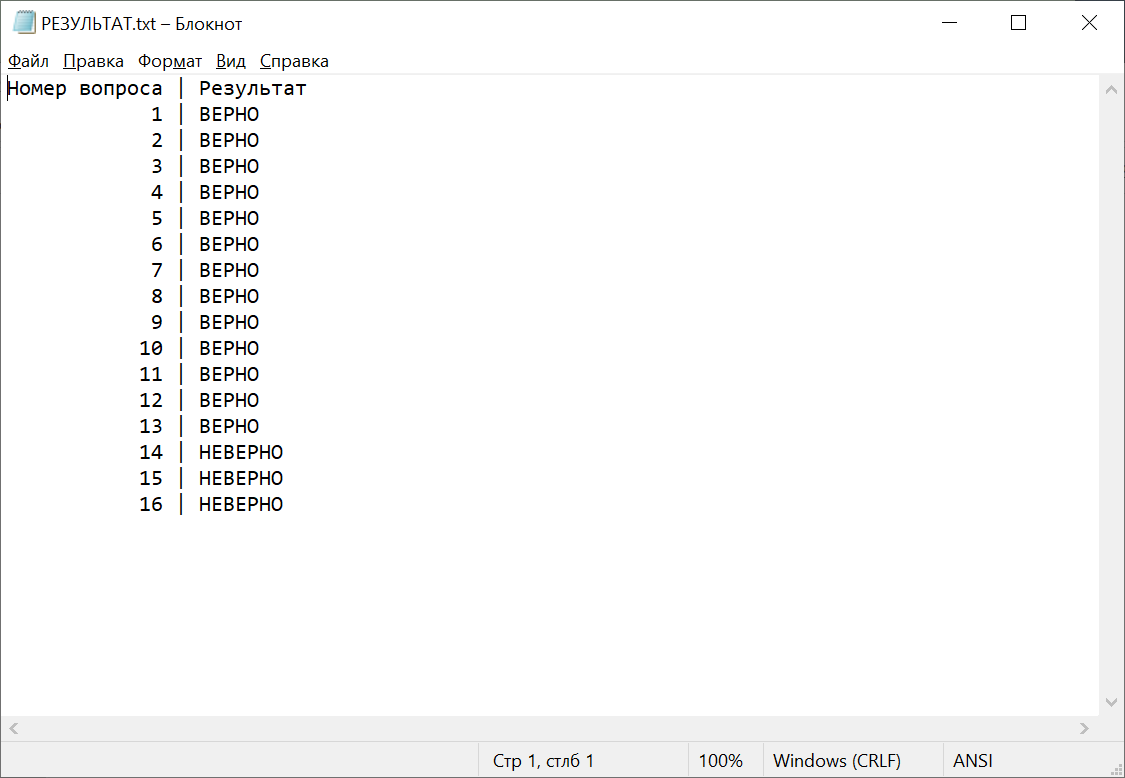


Рисунок 25 — Вывод результатов в текстовый документ

Организация вывода результатов тестирования в документ Microsoft Word происходит с сопутствующим форматированием:

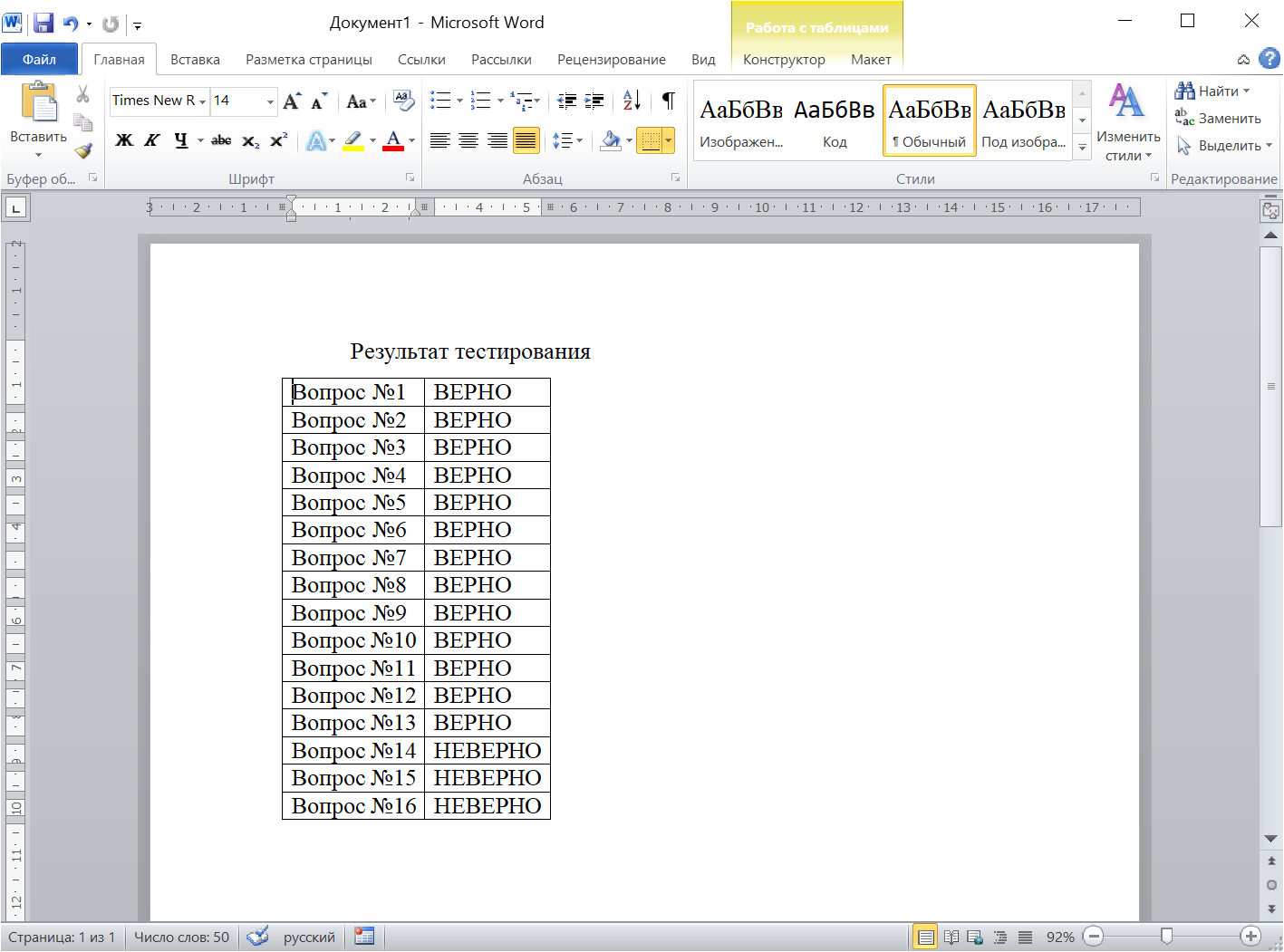


Рисунок 26 — Вывод результатов в документ Microsoft Word

Программный код, реализующий данную возможность:

Код библиотеки:

void CourseworkLibrary::Tools::write\_to\_word(array<bool>^ array)

{

auto word = gcnew WordApplicationClass();

word->Visible = true;

auto missing = Type::Missing;

auto document = word->Documents->Add(missing, missing, missing, missing);

Object^ behavior1 = WdDefaultTableBehavior::wdWord9TableBehavior;

Object^ behavior2 = WdAutoFitBehavior::wdAutoFitContent;

word->Selection->TypeText("Результат тестирования");

Table^ table = word->ActiveDocument->Tables->Add(word->Selection->Range,

16, 2, behavior1, behavior2);

for (size\_t i = 0; i < 16; i++)

{

table->Cell(i + 1, 1)->Range->InsertAfter("Вопрос №" + (i + 1));

String^ str = String::Format("{0:f0}", (array[i] ? "ВЕРНО" : "НЕВЕРНО"));

table->Cell(i + 1, 2)->Range->InsertAfter(str);

}

}

Код программы:

System::Void save\_as\_doc(System::Object^ sender, System::EventArgs^ e)

{

Tools::write\_to\_word(answers);

}

Организация вывода результатов тестирования в лист Microsoft Excel происходит с сопутствующим форматированием:

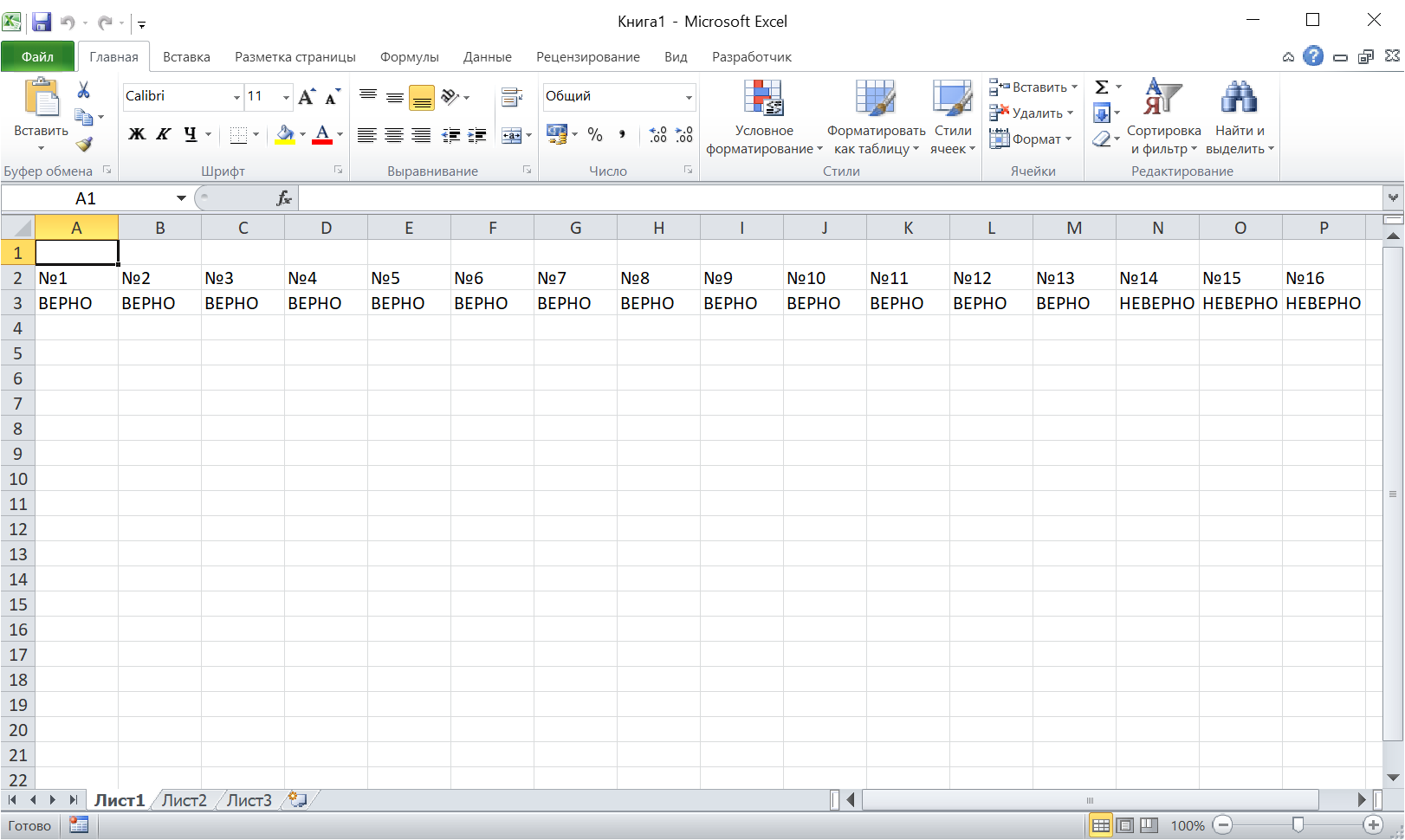


Рисунок 27 — Вывод результатов в лист Microsoft Excel

Программный код, реализующий данную возможность:

Код библиотеки:

void CourseworkLibrary::Tools::write\_to\_excel(array<bool>^ array)

{

auto excel = gcnew ExcelApplication();

excel->Visible = true;

auto missing = Type::Missing;

auto workbook = excel->Workbooks->Add(missing);

auto sheet = workbook->Worksheets;

for (size\_t i = 0; i < 16; i++)

{

excel->Cells[2, i + 1] = "№" + (i + 1);

}

for (size\_t i = 0; i < 16; i++)

{

excel->Cells[3, i + 1] = array[i] ? "ВЕРНО" : "НЕВЕРНО";

}

}

Код программы:

System::Void save\_as\_xls(System::Object^ sender, System::EventArgs^ e)

{

Tools::write\_to\_excel(answers);

}

Организация краткого вывода результатов в виде диаграммы происходит с сопутствующим форматированием (рисунки 21, 22 и 23). Программный код, реализующий данную возможность:

Код библиотеки:

void CourseworkLibrary::Tools::showResults(DataGridView^ view, array<bool>^ results)

{

view->ColumnCount = 2;

view->RowCount = 16;

for (size\_t i = 0; i < 16; i++)

{

view->Rows[i]->Cells[0]->Value = "Вопрос №" + (i + 1);

view->Rows[i]->Cells[1]->Value = results[i] ? "ВЕРНО" : "НЕВЕРНО";

}

int sum = 0;

for (int i = 0; i < view->RowCount; i++)

sum += view->Rows[i]->Height;

view->Height = sum;

view->Columns[0]->AutoSizeMode = DataGridViewAutoSizeColumnMode::AllCells;

view->Columns[1]->AutoSizeMode = DataGridViewAutoSizeColumnMode::AllCells;

}

void CourseworkLibrary::Tools::showResults(Chart^ view, double percentage)

{

view->Titles->Add("РЕЗУЛЬТАТЫ (В ПРОЦЕНТАХ)");

view->Series["Series"]->IsValueShownAsLabel = true;

view->Series["Series"]->Points->AddXY("ВЕРНО", percentage);

view->Series["Series"]->Points->AddXY("НЕВЕРНО", (double)(100 - percentage));

}

Код программы, вызывающий данную функцию, представлен выше.

// Results.h

// Вызывается после загрузки интерфейса

System::Void onLoad(System::Object^ sender, System::EventArgs^ e)

{

label\_title->Text = "На данной странице Вы, " + User::Instance->getName()

", можете ознакомиться с результатами тестирования";

// Получаем процент выполнения

double percentage = UserAnswers::Instance->getPercentage();

// Вывод результатов в элементы

Tools::showResults(result\_dgv, answers);

Tools::showResults(result\_chart, percentage);

// Цвет фона соответствует проценту выполнения

Tools::setBackgroundColor(this, percentage);

}

Реализована графическая заставка, которая появляется перед появлением окна с теоретическим материалом:

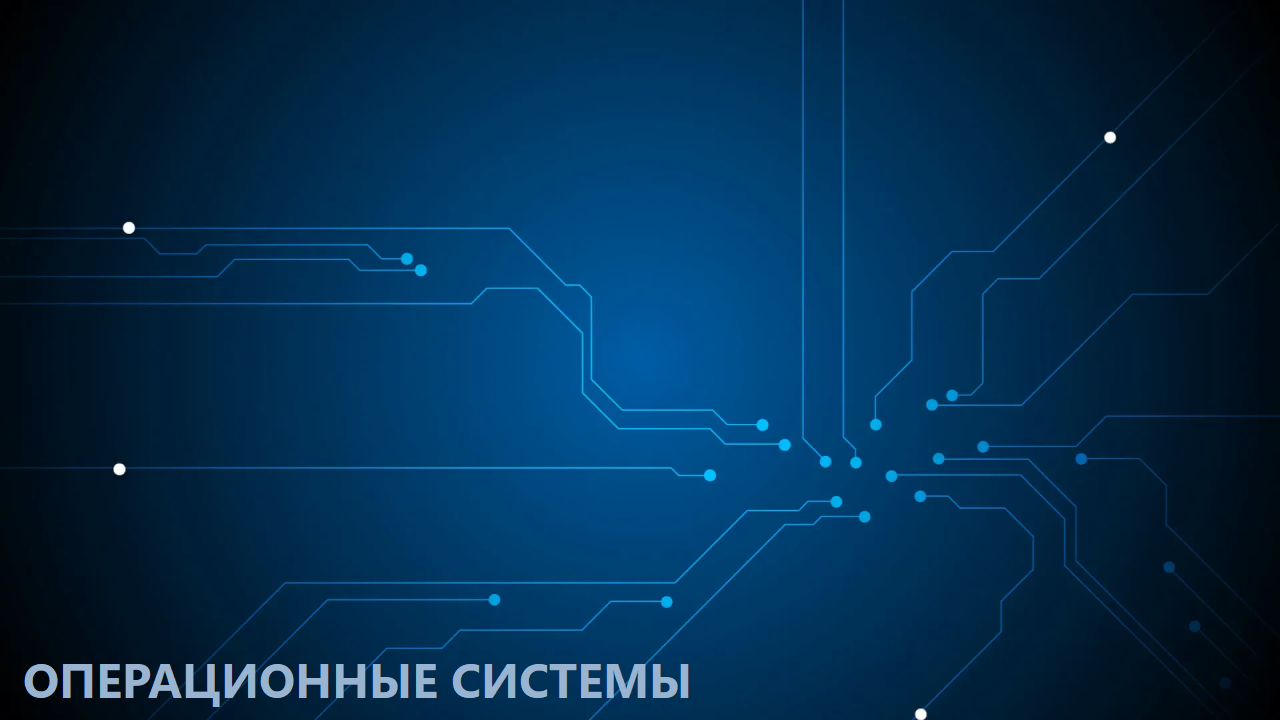


Рисунок 28 — Графическая заставка в приложении

Код программы, реализующий данную возможность:

// Splash.h

// Вызывается после загрузки интерфейса

System::Void onShown(System::Object^ sender, System::EventArgs^ e)

{

timer->Start();

}

// Когда таймер отсчитает 2 сек.

// будет произведён автоматический

// переход на следующую форму

System::Void timer\_tick(System::Object^ sender, System::EventArgs^ e)

{

timer->Stop();

Form^ material = gcnew Material();

this->Hide();

material->Show();

}

Также создан макрос на листе Microsoft Excel, позволяющий применить форматирование на полученных данных в результате тестирования. Макрос запрограммирован на соответствующую кнопку. Результат до форматирования:

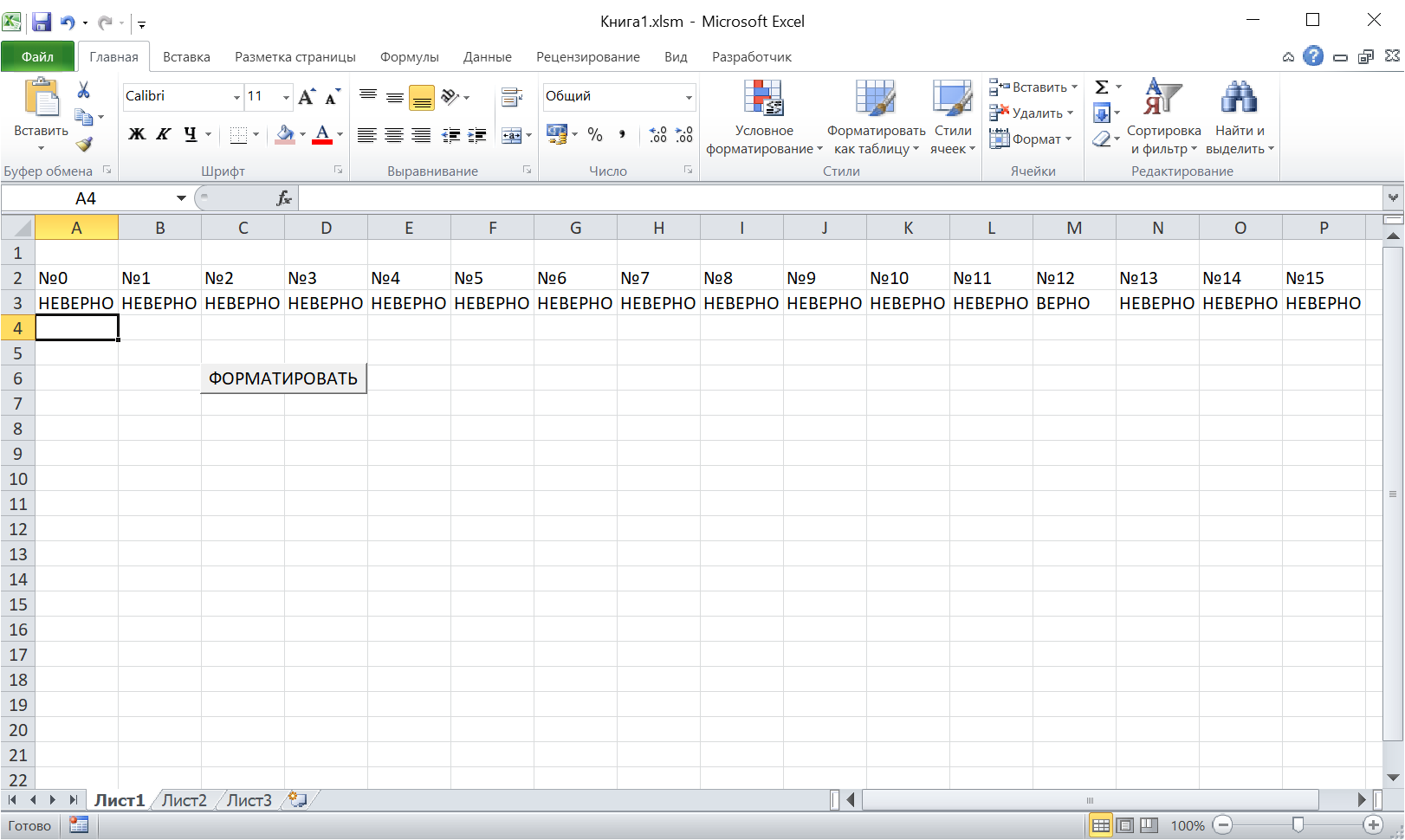


Рисунок 29 — Лист до форматирования

Результат после нажатия кнопки «ФОРМАТИРОВАТЬ»:

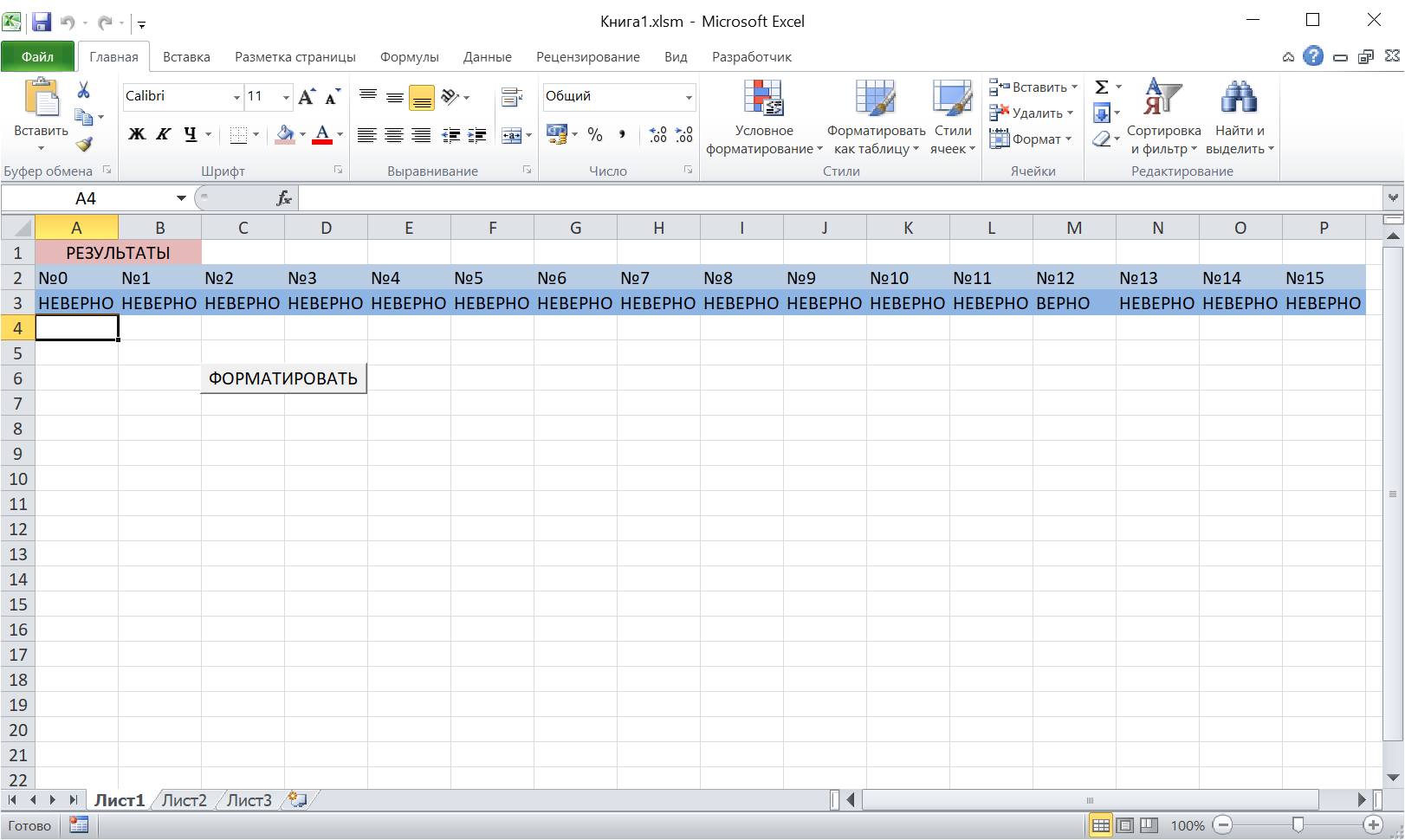


Рисунок 30 — Лист после форматирования

Чтобы сделать архитектуру приложения более гибкой для разработки класс UserAnswers придерживается шаблону проектирования «Одиночка». Таким образом, доступ к публичным методам класса может быть произведён из любой точки программного кода, что обеспечивает удобство разработки с точки зрения объектно-ориентированного программирования.

Код приложения (UserAnswers.h):

#pragma once

public ref class UserAnswers

{

private:

static UserAnswers instance;

array<bool>^ ans = gcnew array<bool>(16);

private:

UserAnswers() {}

UserAnswers(const UserAnswers%);

public:

static property UserAnswers^ Instance

{

UserAnswers^ get()

{

return %instance;

}

}

bool isRight(size\_t index);

void setAnswerResult(size\_t index, bool result);

array<bool>^ getAnswers();

double getPercentage();

};

Код приложения (UserAnswers.cpp):

#include "UserAnswers.h"

UserAnswers::UserAnswers(const UserAnswers%)

{

// Объект пользователя в куче должен быть один

throw gcnew System::InvalidOperationException("Cannot have more than one instances of class");

}

bool UserAnswers::isRight(size\_t index)

{

if (index >= 16) return false;

return ans[index];

}

void UserAnswers::setAnswerResult(size\_t index, bool result)

{

if (index > 16) return;

ans[index - 1] = result;

}

array<bool>^ UserAnswers::getAnswers()

{

return ans;

}

double UserAnswers::getPercentage()

{

double percentage = 0;

for (size\_t i = 0; i < 16; i++)

{

if (ans[i])

percentage += (double)100 / 16;

}

return percentage;

}

По итогам работы над второй главой был разработан электронный конспект и создана программа тестирования знаний по теме «Операционные системы». Были выполнены следующие задачи:

1. разработано электронное пособие и тестовая программа;
2. выведена теоретическая часть с помощью html-файла;
3. разработана DLL-библиотека, содержащая основной функционал программы;
4. организована авторизация пользователя;
5. добавлена возможность сохранять результаты теста в виде текстового файла, базы данных Microsoft Access, документа Microsoft Word и листа Microsoft Excel;
6. обеспечен графический вывод результатов теста в итоговом окне программы.

# ЗАКЛЮЧЕНИЕ

В рамках работы над курсовым проектом был разработан программный продукт, который предназначен для предоставления возможности дистанционного изучения материала по теме курсовой работы, а также для проверки знаний конечного пользователя с помощью тестовой программы.

В результате выполнения работы были решены следующие задачи:

1. проанализирован теоретический материал по теме исследования;
2. разработан программный проект, содержащий электронное пособие и текстовую программу по вышеуказанной теме;
3. оформлен код всех вопросов в виде функций, возвращающих результат;
4. создана DLL-библиотека, содержащая код всех вопросов из предыдущего пункта;
5. разработана система авторизации;
6. организован вывод результатов тестирования в одномерный массив с выводом в DataGridView;
7. организован вывод результатов тестирования в базу Microsoft Access;
8. организован вывод результатов в тестовый документ;
9. организован вывод результатов тестирования в документ Microsoft Word;
10. организован вывод результатов тестирования в лист Microsoft Excel;
11. организован вывод результатов тестирования в одномерный массив;
12. записан макрос, форматирующий лист в Microsoft Excel;
13. организован краткий вывод результатов в виде диаграммы;
14. разработана графическая заставка.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Назаров, С. В. Современные операционные системы : учебное пособие / С. В. Назаров, А. И. Широков. — 3-е изд. — Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 351 c. — ISBN 978-5-4497-0385-9. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/89474.html (дата обращения: 29.10.2020). — Режим доступа: для авторизир. пользователей

2 ГОСТ 7.32-2017. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Отчёт о научно-исследовательской работе. Структура и правила оформления (введён в действие Приказом Росстандарта от 24.10.2017 N 1494-ст) – URL: http://docs.cntd.ru/document/1200157208 (дата обращения: 29.10.2020). – Текст: электронный.

3 ГОСТ 19.201-78. Межгосударственный стандарт. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению (введён в действие Постановлением Госстандарта СССР от 18.12.1978 N 3351) (ред. от 01.06.1981) – URL: http://docs.cntd.ru/document/gost-19-201-78 (дата обращения: 29.10.2020). – Текст: электронный.

# ПРИЛОЖЕНИЕ А

В данном приложении указан список заданий, который предоставляется пользователю в блоке тестирования.

**Вопрос 1**. Возможна ли работа с ЭВМ (работа с прикладными программами), на которой не установлена операционная система?

**Ответ**: нет, невозможна.

**Вопрос 2**. Укажите основные компоненты ЭВМ, без которых запуск оборудования невозможен:

1. жёсткий диск,
2. твердотельный накопитель,
3. ЦПУ,
4. ОЗУ,
5. клавиатура,
6. компьютерная мышь,
7. BIOS.

**Ответ**: 3, 4 и 7.

**Вопрос 3**. В процессе запуска ЭВМ BIOS проверяет физическое наличие необходимых компонентов. Напишите аббревиатуру процесса, о котором идёт речь.

**Ответ**: POST.

**Вопрос 4**. В процессе диагностики BIOS проверяет физическое наличие необходимых компонентов и определяет носитель, с которого будет загружена операционная система. Чтобы диск считался загрузочным, он должен иметь специальную сигнатуру под названием \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. главная управляющая запись;
2. вторичная запоминающая запись;
3. главная загрузочная запись;
4. главная загрузочная подпись.

**Ответ**: Главная загрузочная запись.

**Вопрос 5**. Ядро — центральная часть операционной системы, обеспечивающая приложениям координированный доступ к ресурсам компьютера. Укажите тип ядра, объединяющий достоинства и недостатки монолитного и микроядра.

1. модульное ядро;
2. экзоядро;
3. наноядро;
4. гибридное ядро.

**Ответ**: Гибридное ядро.

**Вопрос 6.** Процессор может работать в двух режимах: пользовательском и привилегированном. Основываясь на знаниях, полученных после прочтения теоретического материала, выберите верные утверждения из списка предложенных:

1. ядро операционной системы работает в пользовательском режиме из соображений безопасности;
2. если программист некорректно использует API (интерфейс, который предоставляет операционная система, для написания программ), то с минимальной вероятностью это может привести к полному отказу системы;
3. неработоспособность ядра может привести к поломке ЭВМ, ведь подобного рода программы имеют непосредственный доступ к компонентам компьютера.

**Ответ:** 2, 3.

**Вопрос 7.** Архитектуры процессоров могут быть 32-х и 64-х битными. Это создаёт ограничение для операционных систем, ведь от этого напрямую зависит количество физических адресов ОЗУ, с которым может работать процессор. Укажите максимально доступный объём оперативной памяти, с которым может работать 32-х разрядная операционная система.

**Ответ:** 4 Гб

**Вопрос 8.** Процесс — абстракция, описывающая выполняющуюся программу, представляющая собой экземпляр выполняемой программы. Основываясь на знаниях, полученных после прочтения теоретического материала, выберите верные утверждения из списка предложенных:

1. концептуально, у каждого процесса есть свой виртуальный CPU;
2. все запущенные процессы выполняются одновременно;
3. многозадачным режимом работы предполагает переключение между существующими процессами, создавая ощущение (псевдо) параллельности работы всех запущенных программ.

**Ответ:** 1, 3

**Вопрос 9.** Концепции процессов и потоков взаимосвязаны. Основываясь на знаниях, полученных после прочтения теоретического материала, выберите верные утверждения из списка предложенных:

1. потоки порождают процессы;
2. процессы порождают потоки;

**Ответ:** 2.

**Вопрос 10.** Контейнер потоков, который занимается поочерёдным выполнением процессорных заданий, называется \_\_\_\_\_ потоков.

**Ответ:** пулом.

**Вопрос 11.** Синхронизация потоков в Windows обеспечивается различными механизмами. Укажите название механизма, о котором было рассказано в теоретическом материале.

**Ответ:** семафор или семафоры

**Вопрос 12.** Соотнесите название концепции с её описанием.

**Ответ:** задание — коллекция процессов, процесс — контейнер для ресурсов, пул потоков — контейнер, занимающийся поочерёдным выполнением заданий.

**Вопрос 13.** Прямая работа с оперативной памятью является наисложнейшим процессом. Чтобы абстрагировать концепцию памяти и не усложнять разработку кода, существует идея \_\_\_\_\_\_\_\_\_\_ памяти.

1. виртуальной;
2. абстрактной.

**Ответ:** 1.

**Вопрос 14.** Со времён создания первых операционных систем инженеры разработали различные механизмы организации работы с памятью В ОС Windows реализована \_\_\_\_\_\_\_\_\_ организация памяти.

1. сегментная;
2. страничная.

**Ответ:** 2.

**Вопрос 15.** Исходя из полученных знаний о работе подсистемы ввода-вывода, выберите верные утверждения из списка предложенных.

1. Грамотная организация параллельной работы устройств ввода-вывода и процессора лежит в основе указанной подсистемы.
2. Согласование скоростей обмена и кэширования данных не предусмотрено системой ввода-вывода.
3. Разделением устройств и данных между процессами занимается система ввода-вывода.
4. Система ввода-вывода способна поддерживать несколько файловых систем.

**Ответ:** 1, 3, 4.

**Вопрос 16.** Для персональных компьютеров операции ввода-вывода могут выполняться тремя способами. Укажите, какой способ наиболее оптимальный на текущий момент развития архитектуры ЭВМ.

1. С помощью программируемого ввода-вывода.
2. С помощью ввода-вывода, управляемого прерываниями.
3. С помощью прямого доступа к памяти.

**Ответ:** 3.

# ПРИЛОЖЕНИЕ Б

Исходный код библиотеки:

// CourseworkLibrary.h

#pragma once

using namespace System;

using namespace System::IO;

using namespace System::Windows::Forms;

using namespace System::Drawing;

using System::Windows::Forms::DataVisualization::Charting::Chart;

using Microsoft::VisualBasic::Interaction;

using System::Data::OleDb::OleDbConnection;

using System::Data::OleDb::OleDbCommand;

using System::Data::OleDb::OleDbException;

using WordApplicationClass = Microsoft::Office::Interop::Word::ApplicationClass;

using Microsoft::Office::Interop::Word::Table;

using Microsoft::Office::Interop::Word::WdDefaultTableBehavior;

using Microsoft::Office::Interop::Word::WdAutoFitBehavior;

using ExcelApplication = Microsoft::Office::Interop::Excel::Application;

namespace CourseworkLibrary {

public ref class Tools

{

public:

static void showInfoBox(String^ message);

static void showErrorBox(String^ message);

static bool isCredentialsValid(TextBox^ field\_username, TextBox^ field\_password);

static bool loadMaterial(WebBrowser^ view, String^ file);

static void handleExitButton(FormClosingEventArgs^ e, String^ message);

static void setHighlighted(Label^ label, bool bold);

static void setBackgroundColor(Form^ form, double percentage);

static bool handleAnswerOne(RadioButton^ answer);

static bool handleAnswerTwo(CheckedListBox^ answer);

static bool handleAnswerThree(TextBox^ answer);

static bool handleAnswerFour(ListBox^ answer);

static bool handleAnswerFive(int answer);

static bool handleAnswerSix(CheckBox^ answer\_1, CheckBox^ answer\_2, CheckBox^ answer\_3);

static bool handleAnswerSeven(int answer);

static bool handleAnswerEight(CheckBox^ answer\_1, CheckBox^ answer\_2, CheckBox^ answer\_3);

static bool handleAnswerNine(RadioButton^ answer);

static bool handleAnswerTen(TextBox^ answer);

static bool handleAnswerEleven(TextBox^ answer);

static bool handleAnswerTwelve(NumericUpDown^ counter\_1, NumericUpDown^ counter\_2, NumericUpDown^ counter\_3);

static bool handleAnswerThirteen(RadioButton^ answer);

static bool handleAnswerFourteen(RadioButton^ answer);

static bool handleAnswerFifteen(CheckBox^ answer\_1, CheckBox^ answer\_2, CheckBox^ answer\_3, CheckBox^ answer\_4);

static bool handleAnswerSixteen(RadioButton^ answer);

static void showResults(DataGridView^ view, array<bool>^ results);

static void showResults(Chart^ view, double percentage);

static void write\_to\_db(array<bool>^ array, String^ timestamp);

static void write\_to\_notepad(array<bool>^ array, String^ timestamp);

static void write\_to\_word(array<bool>^ array);

static void write\_to\_excel(array<bool>^ array);

};

}

// CourseworkLibrary.cpp

#include "CourseworkLibrary.h"

void CourseworkLibrary::Tools::showInfoBox(String^ message)

{

// Вывод информационного сообщения

MessageBox::Show(message, "Информация", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

void CourseworkLibrary::Tools::showErrorBox(String^ message)

{

// Вывод информационного сообщения

MessageBox::Show(message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

static void showRightAnswer(String^ message = "Вы ответили правильно!")

{

// Вывод информационного сообщения

MessageBox::Show(message, "Верно", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

static void showWrongAnswer(String^ message)

{

// Вывод информационного сообщения

MessageBox::Show(message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

static String^ buildFolders(String^ timestamp)

{

// Содание структуры папок

if (!IO::Directory::Exists("results"))

IO::Directory::CreateDirectory("results");

if (!IO::Directory::Exists("results\\" + timestamp))

IO::Directory::CreateDirectory("results\\" + timestamp);

return "results\\" + timestamp;

}

void CourseworkLibrary::Tools::setHighlighted(Label^ label, bool bold)

{

// Выделение полужирным шрифтом

if (bold) label->Font = gcnew Font(label->Font, FontStyle::Bold);

else label->Font = gcnew Font(label->Font, FontStyle::Regular);

}

void CourseworkLibrary::Tools::setBackgroundColor(Form^ form, double percentage)

{

// Цвет итоговой формы зависит от процента выполнения

if (percentage >= 0 && percentage < 33)

{

// RED

form->BackColor = System::Drawing::Color::FromArgb(

static\_cast<System::Int32>(static\_cast<System::Byte>(0xff)),

static\_cast<System::Int32>(static\_cast<System::Byte>(0xe5)),

static\_cast<System::Int32>(static\_cast<System::Byte>(0xe5)));

}

else if (percentage >= 33 && percentage < 66)

{

// YELLOW

form->BackColor = System::Drawing::Color::FromArgb(

static\_cast<System::Int32>(static\_cast<System::Byte>(0xff)),

static\_cast<System::Int32>(static\_cast<System::Byte>(0xff)),

static\_cast<System::Int32>(static\_cast<System::Byte>(0xe5)));

}

else if (percentage >= 66 && percentage <= 100)

{

// GREEN

form->BackColor = System::Drawing::Color::FromArgb(

static\_cast<System::Int32>(static\_cast<System::Byte>(0xe5)),

static\_cast<System::Int32>(static\_cast<System::Byte>(0xff)),

static\_cast<System::Int32>(static\_cast<System::Byte>(0xe5)));

}

}

bool CourseworkLibrary::Tools::isCredentialsValid(TextBox^ field\_username, TextBox^ field\_password)

{

// Проверка валидности имени и пароля

String^ username = field\_username->Text->Trim();

String^ password = field\_password->Text->Trim();

if (username->Equals(""))

{

showErrorBox("Поле имени пользователя не может быть пустым");

return false;

}

if (!password->Equals("password1324"))

{

showErrorBox("Неверный пароль");

return false;

}

return true;

}

bool CourseworkLibrary::Tools::loadMaterial(WebBrowser^ view, String^ file)

{

// Загрузка материала

String^ documentPath = Environment::CurrentDirectory + "\\" + file;

if (!File::Exists(documentPath))

{

showErrorBox("Не найден файл с теоретическим материалом! (" + file + ")\nПереустановите программный продукт");

return false;

}

view->Navigate(documentPath);

return true;

}

void CourseworkLibrary::Tools::handleExitButton(FormClosingEventArgs^ e, String^ message)

{

// Подтверждение выхода из приложения

if (e->CloseReason != CloseReason::UserClosing)

{

Application::Exit();

return;

}

if (MessageBox::Show(message, "Подтверждение", MessageBoxButtons::YesNo,

MessageBoxIcon::Question) == System::Windows::Forms::DialogResult::Yes)

{

Application::Exit();

}

else

{

e->Cancel = true;

}

}

bool CourseworkLibrary::Tools::handleAnswerOne(RadioButton^ answer)

{

bool right = answer->Checked;

if (right) showRightAnswer();

else showErrorBox("Операционная система является узлом основных аппаратных компонентов. " +

"В виду этого, работа ЭВМ без ОС невозможна.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerTwo(CheckedListBox^ answer)

{

bool right =

answer->GetItemChecked(2) &&

answer->GetItemChecked(3) &&

answer->GetItemChecked(6) &&

answer->CheckedIndices->Count == 3;

if (right) showRightAnswer();

else showErrorBox("Запуск ЭВМ невозможен без ЦПУ, ОЗУ и BIOS");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerThree(TextBox^ answer)

{

bool right = answer->Text->Trim()->ToLower()->Equals("post");

if (right) showRightAnswer();

else showErrorBox("При нажатии кнопки питания на устройстве происходит начальная инициализация устройства. " +

"Эта инициализация представляет собой серию диагностик, называемых POST — Power-On Self-Test.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerFour(ListBox^ answer)

{

bool right = answer->SelectedIndex == 2;

if (right) showRightAnswer();

else showErrorBox("Чтобы диск считался загрузочным, он должен иметь " +

"специальную сигнатуру под названием ГЛАВНАЯ ЗАГРУЗОЧНАЯ ЗАПИСЬ");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerFive(int answer)

{

bool right = answer == 4;

if (right) showRightAnswer();

else showErrorBox("Гибридное ядро представляет собой модифицированную версию микроядра, " +

"объединяющую достоинства и недостатки монолитного и микроядра.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerSix(CheckBox^ answer\_1, CheckBox^ answer\_2, CheckBox^ answer\_3)

{

bool right = !answer\_1->Checked && answer\_2->Checked && answer\_3->Checked;

if (right) showRightAnswer();

else showErrorBox("Верными утверждениями являются пункты 2 и 3.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerSeven(int answer)

{

bool right = answer == 3;

if (right) showRightAnswer();

else showErrorBox("Максимально доступный объём оперативной памяти, с которым может работать " +

"32-х разрядная операционная система, ограничен 4-мя гигабайтами.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerEight(CheckBox^ answer\_1, CheckBox^ answer\_2, CheckBox^ answer\_3)

{

bool right = answer\_1->Checked && !answer\_2->Checked && answer\_3->Checked;

if (right) showRightAnswer();

else showErrorBox("Верными утверждениями являются пункты 1 и 3.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerNine(RadioButton^ answer)

{

bool right = answer->Checked;

if (right) showRightAnswer();

else showErrorBox("Процессы порождают потоки.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerTen(TextBox^ answer)

{

bool right = answer->Text->Trim()->ToLower()->Equals("пулом");

if (right) showRightAnswer();

else showErrorBox("Контейнером для процессов называется пул потоков.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerEleven(TextBox^ answer)

{

String^ str = answer->Text->Trim()->ToLower();

bool right = str->Equals("семафор") || str->Equals("семафоры");

if (right) showRightAnswer();

else showErrorBox("Синхронизация потоков в Windows обеспечивается семафорами.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerTwelve(NumericUpDown^ counter\_1, NumericUpDown^ counter\_2, NumericUpDown^ counter\_3)

{

bool right = counter\_1->Value == 2 && counter\_2->Value == 3 && counter\_3->Value == 1;

if (right) showRightAnswer();

else showErrorBox("Задание — коллекция процессов.\nПроцесс — контейнер для ресурсов.\n" +

"Пул потоков — контейнер, занимающийся поочерёдным выполнением заданий.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerThirteen(RadioButton^ answer)

{

bool right = answer->Checked;

if (right) showRightAnswer();

else showErrorBox("Данная концепция называется виртуальной памятью.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerFourteen(RadioButton^ answer)

{

bool right = answer->Checked;

if (right) showRightAnswer();

else showErrorBox("В ОС Windows реализована страничная организация памяти.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerFifteen(

CheckBox^ answer\_1, CheckBox^ answer\_2,

CheckBox^ answer\_3, CheckBox^ answer\_4)

{

bool right = answer\_1->Checked && !answer\_2->Checked && answer\_3->Checked && answer\_4->Checked;

if (right) showRightAnswer();

else showErrorBox("Верными утверждениями являются пункты 1, 3 и 4.");

return right;

}

bool CourseworkLibrary::Tools::handleAnswerSixteen(RadioButton^ answer)

{

bool right = answer->Checked;

if (right) showRightAnswer();

else showErrorBox("В настоящее время в персональных и других компьютерах используется третий способ ввода - вывода.");

return right;

}

void CourseworkLibrary::Tools::showResults(DataGridView^ view, array<bool>^ results)

{

// Вывод в DGV

view->ColumnCount = 2;

view->RowCount = 16;

for (size\_t i = 0; i < 16; i++)

{

view->Rows[i]->Cells[0]->Value = "Вопрос №" + (i + 1);

view->Rows[i]->Cells[1]->Value = results[i] ? "ВЕРНО" : "НЕВЕРНО";

}

int sum = 0;

for (int i = 0; i < view->RowCount; i++)

sum += view->Rows[i]->Height;

view->Height = sum;

view->Columns[0]->AutoSizeMode = DataGridViewAutoSizeColumnMode::AllCells;

view->Columns[1]->AutoSizeMode = DataGridViewAutoSizeColumnMode::AllCells;

}

void CourseworkLibrary::Tools::showResults(Chart^ view, double percentage)

{

// Вывод в диаграмму

view->Titles->Add("РЕЗУЛЬТАТЫ (В ПРОЦЕНТАХ)");

view->Series["Series"]->IsValueShownAsLabel = true;

view->Series["Series"]->Points->AddXY("ВЕРНО", percentage);

view->Series["Series"]->Points->AddXY("НЕВЕРНО", (double)(100 - percentage));

}

static void create\_db(String^ path)

{

// Создание файла БД

ADOX::Catalog^ catalog = gcnew ADOX::CatalogClass();

try

{

catalog->Create("Provider = Microsoft.Jet.OLEDB.4.0; Data Source = " + path);

}

catch (System::Runtime::InteropServices::COMException^ exception)

{

MessageBox::Show(exception->Message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

finally

{

catalog = nullptr;

}

}

static void configure\_db(String^ path)

{

// Создание структуры БД

auto p = gcnew OleDbConnection("Provider = Microsoft.Jet.OLEDB.4.0; Data Source = " + path);

p->Open();

auto c = gcnew OleDbCommand("CREATE TABLE [Результат]([Номер вопроса] counter, [Результат] char(200))", p);

try

{

c->ExecuteNonQuery();

}

catch (OleDbException^ exception)

{

MessageBox::Show(exception->Message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

p->Close();

}

void CourseworkLibrary::Tools::write\_to\_db(array<bool>^ array, String^ timestamp)

{

// Запись в БД

String^ path = buildFolders(timestamp) + "\\РЕЗУЛЬТАТ.mdb";

create\_db(path);

configure\_db(path);

for (size\_t i = 0; i < 16; i++)

{

auto p = gcnew OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Data Source = " + path);

p->Open();

auto c = gcnew OleDbCommand("INSERT INTO[Результат](" +

"[Номер вопроса], [Результат]) VALUES('" + (i + 1) + "','" + (array[i] ? "ВЕРНО" : "НЕВЕРНО") + " ')");

c->Connection = p;

c->ExecuteNonQuery();

}

showInfoBox("Ваши результаты были успешно записаны в таблицу Microsoft Access. Файл отчета может быть найден" +

"в директории results рядом с запускаемым файлом программы.");

}

void CourseworkLibrary::Tools::write\_to\_notepad(array<bool>^ array, String^ timestamp)

{

// Запись в текстовый файл

String^ path = buildFolders(timestamp) + "\\РЕЗУЛЬТАТ.txt";

try {

auto encoding = System::Text::Encoding::GetEncoding(1251);

auto stream = gcnew StreamWriter(path, false, encoding);

stream->WriteLine("Номер вопроса | Результат");

for (int i = 0; i < 16; i++)

{

stream->WriteLine(String::Format("{0, 16}", (i + 1) + " | ") + (array[i] ? "ВЕРНО" : "НЕВЕРНО"));

}

stream->Close();

showInfoBox("Ваши результаты были успешно записаны в тестовый файл. Файл отчета может быть найден " +

"в директории results рядом с запускаемым файлом программы.");

}

catch (System::Exception^ exception) {

MessageBox::Show(exception->Message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

void CourseworkLibrary::Tools::write\_to\_word(array<bool>^ array)

{

// Запись в документ Word

auto word = gcnew WordApplicationClass();

word->Visible = true;

auto missing = Type::Missing;

auto document = word->Documents->Add(missing, missing, missing, missing);

Object^ behavior1 = WdDefaultTableBehavior::wdWord9TableBehavior;

Object^ behavior2 = WdAutoFitBehavior::wdAutoFitContent;

word->Selection->TypeText("Результат тестирования");

Table^ table = word->ActiveDocument->Tables->Add(word->Selection->Range,

16, 2, behavior1, behavior2);

for (size\_t i = 0; i < 16; i++)

{

table->Cell(i + 1, 1)->Range->InsertAfter("Вопрос №" + (i + 1));

String^ str = String::Format("{0:f0}", (array[i] ? "ВЕРНО" : "НЕВЕРНО"));

table->Cell(i + 1, 2)->Range->InsertAfter(str);

}

}

void CourseworkLibrary::Tools::write\_to\_excel(array<bool>^ array)

{

// Запись в лист Excel

auto excel = gcnew ExcelApplication();

excel->Visible = true;

auto missing = Type::Missing;

auto workbook = excel->Workbooks->Add(missing);

auto sheet = workbook->Worksheets;

for (size\_t i = 0; i < 16; i++)

{

excel->Cells[2, i + 1] = "№" + (i + 1);

}

for (size\_t i = 0; i < 16; i++)

{

excel->Cells[3, i + 1] = array[i] ? "ВЕРНО" : "НЕВЕРНО";

}

}

Код проверки авторизации:

System::Void button\_start\_click(System::Object^ sender, System::EventArgs^ e)

{

String^ username = field\_username->Text->Trim();

if (Tools::isCredentialsValid(field\_username, field\_password))

{

User^ user = User::Instance;

// Проверяем, правильно ли пользователь ввел свои данные второй раз

if (!user->checkAuth(username))

{

Tools::showErrorBox("Введенное имя пользователя не совпадает с именем, зарегистрированным " +

"ранее!\nПроверьте корректность ввода.");

return;

}

User::Instance->logIn(username);

Tools::showInfoBox("Аутентификация выполнена успешно!");

parent->Hide();

this->Hide();

next->Show();

}

}