|  |
| --- |
| Харьковский национальный университет радиоэлектроники |
| (полное название высшего ученого заведения) |
| Кафедра «Электронных вычислительных машин» |
| (полное название кафедры) |

**КУРСОВОЙ ПРОЕКТ**

**(РАБОТА)**

|  |  |
| --- | --- |
| по курсу: | Системное программное обеспечение |
|  | (название дисциплины) |
| на тему: | Разработка программного обеспечение для управления |
|  | запущенными на локальном и удаленном компьютере процессами. |
|  |  |
|  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Студентки | 3 | | | курса | | КИ-15-4 | группы |
|  | направления подготовки | | | | Компьютерная инженерия и управление | | | |
|  | специальность | | Компьютерная инженерия | | | | | |
|  | Розенберг М. Э. | | | | | | | |
|  | (фамилия и инициалы) | | | | | | | |
|  | Руководитель |  | | | | | | |
|  |  | | | | | | | |
|  | (должность, ученое звание, научная степень, фамилия и инициалы) | | | | | | | |
|  | Национальная шкала | | | |  | | | |
|  | Количество балов: | | |  | | Оценка: ECTS | |  |
|  |  | | | | | | | |

г. Харьков – 2017 год

Харьковский национальный университет радиоэлектроники

(наименование высшего учебного заведения)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кафедра: | Электронных вычислительных машин | | | | |
| Дисциплина: | Системное программное обеспечение | | | | |
| Специальность: | Компьютерная инженерия | | | | |
| Курс: | 3 | Группа: | КИ-15-4 | Семестр: | 5 |

**З А Д А Н И Е**

на курсовой проект (работу) студента

Розенберг Мария Эдуардовна

(фамилия, имя, отчество)

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Тема проекта (работы) | Разработка программное обеспечение для управления | | |
| запущенными на локальном и удаленном компьютере процессами | | | |
| 2. Срок сдачи студентом законченного проекта (работы) | | |  |
| 3. Исходные данные к проекту (работе) | | Разработать программное обеспечение | |
| для управления запущенными на локальном и удаленном компьютере процессами. | | | |
| Программное обеспечение предоставляет возможности по просмотру списка | | | |
| запущенных процессов, созданию новых и удаления (остановки) выполняемых. | | | |
| Пользователь может задать произвольное количество удаленных компьютеров, за | | | |
| которыми производится наблюдение. Для каждого компьютера пользователь может | | | |
| определить список «запрещенных» процессов, которые останавливаются сразу как | | | |
| только обнаруживаются. | | | |
| 4. Содержание курсовой работы : | | | |
| Разработка механизма взаимодействие с процессами на локальном и удаленном | | | |
| компьютере. Разработка графического пользовательского интерфейса, с | | | |
| обеспечивающего взаимодействие пользователем. | | | |
| 5. Дата выдачи задания: |  | | |

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № этапа | Содержание выполняемых работ | Период | Отметка |
| 1 | Выдача заданий | 2-я неделя |  |
| 2 | Ознакомление с литературными источниками, анализ и выбор метода решения поставленной задачи | 3-4 неделя |  |
| 3 | Разработка алгоритмов решения, выбор системных средств решения задач курсового проектирования | 5-6 неделя |  |
| 4 | Проектирование приложения | 7-11 неделя |  |
| 5 | Отладка и тестирование приложения | 12-14 неделя |  |
| 6 | Оформление пояснительной записки | 15 неделя |  |
| 7 | Защита курсового проекта | 16-17 неделя |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

РЕФЕРАТ

Пояснительная записка содержит 29 страниц, 4 рисунка, 4 раздела, 1 приложение, 5 источников литературы.

Цель работы — разработать программное обеспечение для управления запущенными на локальном и удаленном компьютере процессами.

Метод исследования – изучение литературы, написание и отладка программы на компьютере.

Программа написана на языке C# в среде разработки Visual Studio 2015.

ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ, VISUAL STUDIO 2015, ОПЕРАЦИОННАЯ СИСТЕМА, C#, WMI, WINDOWS FORMS.

Оглавление

[Введение 6](#_Toc482575894)

[1 Анализ предметной области и постановка задач 8](#_Toc482575895)

[2 Разработка структуры и алгоритма программы 10](#_Toc482575896)

[3 Описание программной реализации 13](#_Toc482575897)

[4 Инструкция пользователя 16](#_Toc482575898)

[Выводы 19](#_Toc482575899)

[Список литературы 20](#_Toc482575900)

[Приложение А 21](#_Toc482575901)

# Введение

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами. Процесс (или по-другому, задача) - абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами. Обычно при загрузке ОС создаются несколько процессов. Некоторые из них являются высокоприоритетными процессами, обеспечивающими взаимодействие с пользователями и выполняющими заданную работу. Остальные процессы являются фоновыми, они не связаны с конкретными пользователями, но выполняют особые функции – например, связанные с электронной почтой, Web-страницами, выводом на печать, передачей файлов по сети, периодическим запуском программ (например, дефрагментации дисков) и т.д.

Диспетчер задач — компьютерная программа (утилита) для вывода на экран списка запущенных процессов и потребляемых ими ресурсов. В качестве дополнительных функций, диспетчер задач может предложить возможность завершить один из процессов или присвоить ему другой приоритет. На данный момент сложно представить использование компьютера без диспетчера задач, так как очень часто возникает необходимость просмотреть список запущенных в данный момент задач, создать новую задачу или остановить ненужную задачу. Windows Task Manager в Windows NT можно вызвать, одновременно нажав клавиши Ctrl+Shift+Esc. В Windows NT и в Windows XP существует более известная комбинация клавиш — Ctrl+Alt+Del. Диспетчер задач можно также запустить в командной строке, введя имя его исполняемого файла (taskmgr.exe) или выбрав соответствующий пункт в контекстном меню панели задач.

Однако встроенная утилита позволяет просматривать только задачи, запущенные на данном компьютере. Это становится не очень удобным, когда необходимо управлять сразу несколькими компьютерами и следить за их работой и запускать там новые задачи.

Поэтому целью данной курсовой работы, является описание разработки приложения, которое позволит пользователю управлять процессами как на своем локальном компьютере, так и на любом другом удаленном компьютере, к которому у него есть доступ. Данное приложение является довольно актуальным так как существует довольно мало приложений способных с управлять процессами на удаленном компьютере, а так же следить за списком запущенных задач и закрывать «запрещенные» приложения.

# 1 Анализ предметной области и постановка задач

В данной работе рассматривается вопрос создания программы «Часы», в которой есть возможность просматривать текущее время.

С целью максимизации удобности работы программы, была добавлена возможность ускорения мяча.

Сферой использования программы может быть, как обычное развлечение, так и исследование игровой индустрии, которая являлась актуальной в прошлом или для того, чтоб вспомнить классические игры.

Разрабатываемое приложение может быть использовано любым пользователем компьютера для просмотра времени.

В результате выполнения данной курсовой работы программа должна обеспечивать следующий функционал:

* изменять размеры окна с часами;
* изменять тип часов на цифровые или стрелочные;
* изменять цвет фона часов;
* изменять цвет стрелок и цифр на часах.

Для разработки данной программы, были использованы стандартные средства библиотек Windows. Для реализации окна, были реализованы 4 основных функции, которые необходимы для реализации Windows-приложений. Главная функция «WinMain» в которой вызываются функции регистрации класса окна и функция его создания, а также реализован цикл обработки сообщений, функция регистрации класса окна «MyRegisterClass», функция «InitInstance», которая сохраняет обработки экземпляра и создает главное окно и так же функция «WndProc», в которой и будет выполнятся реализация программы «Часы». Реализацию этих основных функций создания окна, можно посмотреть в приложении А. В программе также были использованы принципы объектно-ориентированного программирования, для удобства представление объектов. Для реализации движения стрелок был использован стандартный таймер, который создается с помощью функции «SetTimer», а уничтожается с помощью функции «KillTimer», использование этих функций можно увидеть в приложении A.

В основе всей программы, лежит вполне базовая, не сложная реализация, которая состоит из множества условий, циклов и переключателей.

Так же в программе присутствуют ресурсы, такие как иконка, меню, диалоговое окно. Реализация выбора в меню происходит в оконной процедуре, при получении сообщения «WM\_COMMAND». В меню есть несколько вкладок, «Файл», «Справка», «Тип часов». Для диалогового окна, которое находится в вкладке «О программе…» реализована функция «About».

Для отображения всплывающих уведомлений или сообщений, используются диалоговые окна, создаваемые с помощью функции «MessageBox».

# 2 Разработка структуры и алгоритма программы

Структура данной программы, совпадает с структурой Windows-приложений, в программе реализованы 4 основные функции, главная функция «WinMain» в которой вызываются функции регистрации класса окна и функция его создания, а также реализован цикл обработки сообщеий, функция регистрации класса окна «MyRegisterClass», функция «InitInstance», которая сохраняет обработки экземпляра и создает главное окно и так же функция «WndProc». Так же, существует дополнительная функция, которая реализовывает диалоговое окно. Для более хорошего структурирования программы, был использован объектно-ориентированный подход, а именно созданы классы для точек, стрелок и часов, которые хранят в себе все необходимые данные. Так как часов в данной программе было разработано несколько типов, то для правильного их использования в коде и простого добавления новых типов часов был создан базовый класс для них. Данный класс должны наследовать все другие типы часов для использования в программе.

Так же, для более удобной отладки и использования программного кода, программа разбита на отдельные файлы:

* Ticker.cpp — файл который содержит основные функции программы;
* point.h — содержит все структуру для хранения координат точек;
* arrow.h — файл, содержащий класс для хранения информации о стрелках и вспомогательных методов для расчета положения стрелок;
* clock.h — содержит базофую информацию о часах;
* arrowClock.h — файл, для работы со стрелочными часами;
* digitClock.h — файл, для работы с цифровыми часами;
* resource.h — файл, в котором хранятся идентификаторы используемых ресурсов.

Основная обработка действий программы, происходит в оконной процедуре, которая получает различные сообщения, в нашем случае это: «WM\_TIMER», «WM\_COMMAND», «WM\_PAINT», «WM\_DESTROY». Действия, выполняемые при получении этих сообщений, будут описаны ниже в алгоритме программы, а также в описании программной реализации.

В разработанной программе используется базовый, не сложный алгоритм:

* Для начала создается окно, в котором и будет происходить игровой процесс, с помощью основных функций стандартной библиотеки, про которые мы говорили выше.
* После этого, в оконной процедуре программы объявляются основные переменные, а также создаются часы, выставляются цвета по умолчанию.
* Далее, программа обрабатывает сообщение «WM\_PAINT» в котором создаются необходимые кисти, рисуется циферблат, различные стрелки, выставляется цвет фона и элементов часов.
* Для того, чтоб стрелки начали двигаться, необходимо запустить таймер, который автоматически запускается после запуска программы.
* После запуска таймера, оконной процедуре посылается сообщение «WM\_TIMER» в которой находится расчет координат стрелок часов для текущего времени и далее выполняется функция «RedrawWindow» в которую передаются координаты перерисовки, функция делает выбранную область недействительной, что заставляет программу заново перерисовывать изображение часов.
* Когда пользователь закрывает программу, с помощью меню или с помощью кнопки закрытия в правом верхнем углу, оконной процедуре посылается сообщение «WM\_DESTROY» и происходит закрытие окна.

В разработанной программе присутствует несложные алгоритмы для расчета положения стрелок часов в каждый момент времени. Для этих расчетов необходимо было находить положение точки на кругу, по углу поворота относительно вертикальной прямой. Тогда координату х необходимой точки можно найти как произведение длины отрезка (радиуса круга) на синус угла поворота, а координату y — как произведение длины отрезка на косинус угла поворота.

Так же по заданному времени необходимо было рассчитывать угол поворота стрелки на часах. Для этого было высчитано количество градусов за которое стрелка должна поворачиваться за час, минуту или секунду:

* стрелка для часов за каждый час поворачивается на 30 градусов и каждой минуту на половину градуса;
* минутная стрелка поворачивается на 6 градусов каждую минуту и на 0.1 градуса каждую секунду;
* секундная стрелка каждую секунду поворачивается на 6 градусов.

# 3 Описание программной реализации

Программа, которая была создана, при выполнении курсового проекта, создавалась в среде разработки Visual Studio 2015, средствами WinApi, на языке С++.

Для удобства, программа разбита на несколько файлов: Ticker.cpp – основной файл программы, основные функции которого мы рассмотрели в разделе 2, так же дополнительные файлы объектов: Point.h, Arrow.h, Clock.h, ArrowClock.h, DigitClock.h и файл ресурсов resource.h, которые содержат в себе необходимые для работы программы модули, такие как: классы и их методы, идентификаторы ресурсов.

Для начала рассмотрим созданные в программе для удобства дополнительные файлы классов объектов:

Файл Point.h содержит структуру для хранения координат любых точек. В нем присутствует два поля х и у, а также конструктор с двумя параметрами для создания точки.

struct Point

{

Point(double x, double y) : x(x), y(y) { }

double x;

double y;

};

Пример 3.1

Файл Arrow.h содержит информацию о стрелках, а именно их длину и ширину. И так же он содержит вспомогательный метод для расчета положения стрелки в зависимости от ее угла поворота.

class Arrow

{

public:

Arrow(double length, int width);

Point GetPosition(int size, int angle);

double GetLength(int size);

int GetWidth();

private:

double length;

int width;

};

Пример 3.2

Класс Arrow можно создать с помощью конструктора, принимающего относительный размер стрелки и ее ширину. Для получения этих данных можно использовать методы GetLength и GetWidth соответственно.

Файл Clock.h содержит базовый класс для всех часов и базовые методы, необходимые в системе для работы с часами.

class Clock

{

public:

Clock();

virtual ~Clock();

virtual void Draw(HDC hdc, RECT rect, DWORD color);

protected:

tm GetNow();

int GetCenter(RECT rect);

};

Пример 3.3

Данный класс содержит виртуальную функцию для рисования часов на экране. Каждый наследник этого класса переопределяет ее и реализует свой уникальный способ для отображения. Так же класс Clock содержит несколько вспомогательных функций для получения текущего времени (GetNow) и для получения координат центра окна приложения.

Класс Clock имеет двух наследников — ArrowClock и DigitClock.

ArrowClock состоит из массива стрелок и набора вспомогательных функций для отображения стрелок на экране и функций для рисования циферблата.

DigitClock переопределяет функцию Draw и с помощью встроенных функций WinApi отображает форматированное представление текущего времени.

Так же было использовано стандартное диалоговое окно Windows для выбора цвета. Для этого была разработана функция OpenColorChooseDialog, которая подготавливает необходимые параметры для вызова стандартной функции ChooseColor и потом отображает пользователю окно с палитрой из разных цветов и предлагает выбрать нужные, после чего функция возвращает выбранный пользователем цвет.

DWORD OpenColorChooseDialog(HWND hwnd, DWORD currentColor)

{

CHOOSECOLOR cc;

COLORREF acrCustClr[16];

ZeroMemory(&cc, sizeof(cc));

cc.lStructSize = sizeof(cc);

cc.hwndOwner = hwnd;

cc.lpCustColors = (LPDWORD)acrCustClr;

cc.rgbResult = currentColor;

cc.Flags = CC\_FULLOPEN | CC\_RGBINIT;

if (ChooseColor(&cc) == TRUE)

{

return cc.rgbResult;

}

return currentColor;

}

Пример 3.4

# 4 Инструкция пользователя

Для корректной работы программы не нужно делать никаких дополнительных действий. После запуска приложения оно автоматически начнет свою работу. При запуске по умолчанию выбирается стрелочный вид часов (см. рис. 4.1).

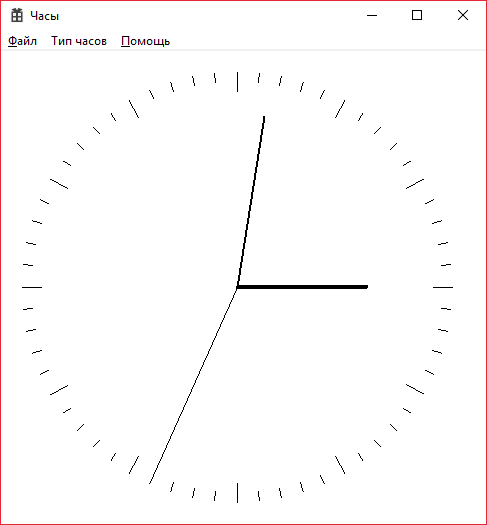


Рисунок 4.1 — Вид стрелочных часов

Данной отображение можно изменить с помощью меню на цифровые часы. Для этого необходимо в меню «Тип часов» выбрать пункт «Цифровые» и изображение окна автоматически замениться на новый тип (см. рис. 4.2). Аналогично можно переключиться обратно на стрелочные часы, выбрав в том же меню пункт «Стрелочные».

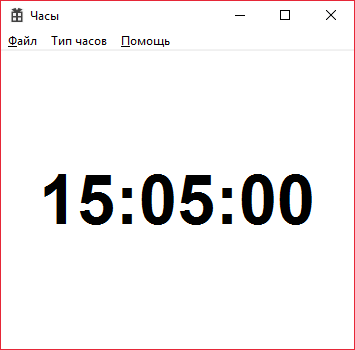


Рисунок 4.2 — Цифровые часы

При желании пользователь может изменить цвет фона и элементов часов с помощью нескольких кликов. Для этого необходимо перейти в пункт меню «Файл» и выбрать интересующий элемент, цвет которого необходимо изменить. После этого пользователю откроется стандартное диалоговое окно Windows для выбора цвета (см. рис. 4.3). После выбора цвета пользователем он автоматически примениться для всех типов часов.

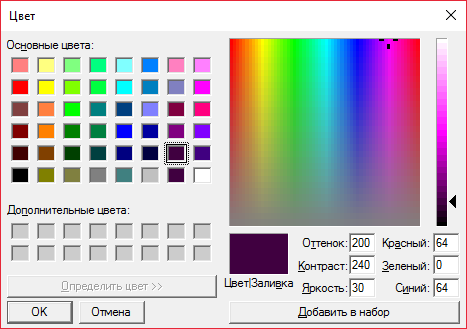


Рисунок 4.3 — Выбор цвета

Так же, пользователь может изменять размеры экрана с помощью мышки. При изменении окна часы будут автоматически перестраиваться под новые размеры (см рис. 4.4).

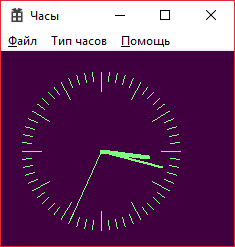


Рисунок 4.4 — Изменение размеров окна

Аналогично для цифровых часов, они будут подстраиваться под размеры окна и всегда отображать по центру экрана.

# Выводы

В ходе выполнения данной работы в среде Visual Studio 2015 с помощью языка программирования C++ и средств WinApi была разработана программа, которая позволяет узнавать текущее время.

Использование данной программы, дает возможность отвлечься, развлечься, вспомнить классические игры или исследовать развитие игровой индустрии, от прошлого к настоящему.

Программа проста в использовании, так как в ней интуитивно понятный интерфейс, так же была составлена для пользователя инструкция использования программы.

Разработанный код программы был разбит на несколько файловых блоков, по принятому стилю программирования. Создан основной блок выполнения программы, а также дополнительные блоки, которые содержат в себе реализацию описания объектов программы, их методы и их реализацию, также создан дополнительный файл, который хранит в себе идентификаторы ресурсов.

На данном этапе проектирования, функционал программы логически завершен. При продолжении работы над данным проектом, в сторону расширения функционала, возможно добавление различных функций, а именно: синхронизация времени с интернетом, возможность добавление будильников, таймеров или секундомеров, добавление календаря и возможности создание заметок и событий.

В своей структуре программа не имеет сложных алгоритмов, использованные функции легкие в использовании, к тому же, функции стандартной библиотеки Windows подробно описаны во многих источниках. Это позволяет без особых проблем написать или отредактировать код программы человеку с базовыми знаниями языка С++.

# Список литературы

1. Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е изд. [Текст]: пер. с англ.- М.: Вильямс, 2013 – 1312 с.
2. Уотсон, К., Нейгел, К., Педерсен, Я. Visual C# 2010: полный курс. [Текст] : Пер. с англ. – М.:ООО «И.Д. ильямс», 2011 – 960 с.
3. Шилдт Г. C# 4.0.: Полное руководство [Текст]: пер. с англ.- М.: Вильямс, 2011 – 1056 с.
4. Попов, А., Шикин, Е. Администрирование Windows с помощью WMI и WMIC [Текст]: 2004 – 752 с.
5. Петцольд, Ч., Программирование с использованием Microsoft Windows Forms [Текст]: 2006 – 433 с.

# Приложение А

Файл Ticker.cpp

// Ticker.cpp : Defines the entry point for the application.

//

#include "stdafx.h"

#include "Ticker.h"

#include "Clock.h"

#include "ArrowClock.h"

#include "DigitClock.h"

#include <windows.h>

#include <Commdlg.h>

#define MAX\_LOADSTRING 100

// Global Variables:

HINSTANCE hInst; // current instance

WCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

WCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

Clock\* ticker;

DWORD elementColor;

DWORD backgroundColor = 0xFFFFFF;

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

ticker = new ArrowClock();

// Initialize global strings

LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadString(hInstance, IDC\_TICKER, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance(hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_TICKER));

MSG msg;

// Main message loop:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_TICKER));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_TICKER);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Store instance handle in our global variable

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

SetTimer(hWnd, 1, 1000, NULL);

return TRUE;

}

DWORD OpenColorChooseDialog(HWND hwnd, DWORD currentColor)

{

CHOOSECOLOR cc;

COLORREF acrCustClr[16];

ZeroMemory(&cc, sizeof(cc));

cc.lStructSize = sizeof(cc);

cc.hwndOwner = hwnd;

cc.lpCustColors = (LPDWORD)acrCustClr;

cc.rgbResult = currentColor;

cc.Flags = CC\_FULLOPEN | CC\_RGBINIT;

if (ChooseColor(&cc) == TRUE)

{

return cc.rgbResult;

}

return currentColor;

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Parse the menu selections:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

case ID\_ARROWCLOCK:

delete ticker;

ticker = new ArrowClock();

break;

case ID\_DIGITCLOCK:

delete ticker;

ticker = new DigitClock();

break;

case ID\_BACKGROUNDCOLOR:

backgroundColor = OpenColorChooseDialog(hWnd, backgroundColor);

break;

case ID\_CLOCKCOLOR:

elementColor = OpenColorChooseDialog(hWnd, elementColor);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

RECT rect;

HDC hdc = BeginPaint(hWnd, &ps);

GetClientRect(hWnd, &rect);

FillRect(hdc, &rect, CreateSolidBrush(backgroundColor));

SetBkColor(hdc, backgroundColor);

ticker->Draw(hdc, rect, elementColor);

EndPaint(hWnd, &ps);

}

break;

case WM\_TIMER:

RedrawWindow(hWnd, NULL, NULL, RDW\_INVALIDATE);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

// Message handler for about box.

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

Файл Point.h

#pragma once

struct Point

{

Point(double x, double y) : x(x), y(y) { }

double x;

double y;

};

Файл Arrow.cpp

#include "stdafx.h"

#define \_USE\_MATH\_DEFINES

#include "Arrow.h"

#include <math.h>

Arrow::Arrow(double length, int width)

: length(length), width(width)

{

}

Point Arrow::GetPosition(int size, int angle)

{

double radian = angle \* M\_PI / 180;

double x = GetLength(size) \* sin(radian);

double y = GetLength(size) \* cos(radian);

return Point(x,y);

}

double Arrow::GetLength(int size)

{

return length\*size;

}

int Arrow::GetWidth()

{

return width;

}

Файл Clock.cpp

#include "stdafx.h"

#include "Clock.h"

Clock::Clock()

{

}

Clock::~Clock()

{

}

void Clock::Draw(HDC hdc, RECT rect, DWORD color)

{

}

tm Clock::GetNow()

{

time\_t timev;

time(&timev);

struct tm now;

localtime\_s(&now, &timev);

return now;

}

int Clock::GetCenter(RECT rect)

{

return max(min(rect.right, rect.bottom), 200) / 2;

}

Файл ArrowClock.h

#pragma once

#include "Clock.h"

#include "Arrow.h"

class ArrowClock :

public Clock

{

public:

ArrowClock();

~ArrowClock() override;

void Draw(HDC hdc, RECT rect, DWORD color) override;

void DrawBackground(HDC hdc, Point center);

private:

Arrow \_arrows[3];

};

Файл ArrowClock.cpp

#include "stdafx.h"

#include "ArrowClock.h"

int radius = 200;

ArrowClock::ArrowClock() : \_arrows{

Arrow(1, 1),

Arrow(0.8, 2),

Arrow(0.6, 4)

}

{

}

ArrowClock::~ArrowClock()

{

}

void DrawLine(HDC hdc, Point start, Point end)

{

MoveToEx(hdc, start.x, start.y, NULL);

LineTo(hdc, end.x, end.y);

}

Point NormalizePoint(Point center, Point p)

{

return Point(p.x + center.x, -p.y + center.y);

}

void ArrowClock::DrawBackground(HDC hdc, Point center)

{

for (int i = 0; i < 360; i += 6)

{

int length = 10;

if (i % 5 == 0)

{

length \*= 2;

}

DrawLine(

hdc,

NormalizePoint(center, \_arrows[0].GetPosition(radius - length, i)),

NormalizePoint(center, \_arrows[0].GetPosition(radius, i)));

}

}

void DrawArrow(HDC hdc, Arrow arrow, Point center, double angle, DWORD color)

{

BeginPath(hdc);

HPEN pen = CreatePen(PS\_SOLID, arrow.GetWidth(), color);

SelectObject(hdc, pen);

DrawLine(hdc, center, NormalizePoint(center, arrow.GetPosition(radius, angle)));

DeleteObject(pen);

EndPath(hdc);

StrokePath(hdc);

}

void DrawHourArrow(HDC hdc, Arrow arrow, Point center, DWORD color, tm now)

{

DrawArrow(hdc, arrow, center, now.tm\_hour \* 30 + now.tm\_min\*0.5, color);

}

void DrawMinuteArrow(HDC hdc, Arrow arrow, Point center, DWORD color, tm now)

{

DrawArrow(hdc, arrow, center, now.tm\_min \* 6 + now.tm\_sec\*0.1, color);

}

void DrawSecondArrow(HDC hdc, Arrow arrow, Point center, DWORD color, tm now)

{

DrawArrow(hdc, arrow, center, now.tm\_sec \* 6, color);

}

void ArrowClock::Draw(HDC hdc, RECT rect, DWORD color)

{

BeginPath(hdc);

HPEN pen = CreatePen(PS\_SOLID, 1, color);

SelectObject(hdc, pen);

int size = GetCenter(rect);

radius = size - 20;

Point center = Point(size, size);

DrawBackground(hdc, center);

DeleteObject(pen);

EndPath(hdc);

StrokePath(hdc);

tm now = GetNow();

DrawHourArrow(hdc, \_arrows[2], center, color, now);

DrawMinuteArrow(hdc, \_arrows[1], center, color, now);

DrawSecondArrow(hdc, \_arrows[0], center, color, now);

}

Файл DigitClock.h

#pragma once

#include "Clock.h"

class DigitClock :

public Clock

{

public:

DigitClock();

~DigitClock() override;

void Draw(HDC hdc, RECT rect, DWORD color) override;

};

Файл DigitClock.cpp

#include "stdafx.h"

#include "DigitClock.h"

DigitClock::DigitClock()

{

}

DigitClock::~DigitClock()

{

}

void DigitClock::Draw(HDC hdc, RECT rect, DWORD color)

{

tm now = GetNow();

WCHAR buffer[10];

SetTextColor(hdc, color);

HFONT hFont = CreateFont(80, 0, 0, 0, FW\_BOLD, 0, 0, 0, 0, 0, 0, 2, 0, \_T("SYSTEM\_FIXED\_FONT"));

HFONT hTmp = (HFONT)SelectObject(hdc, hFont);

SetBkMode(hdc, TRANSPARENT);

DrawText(hdc, buffer, wsprintf(buffer, \_T("%02d:%02d:%02d"), now.tm\_hour, now.tm\_min, now.tm\_sec), &rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);

DeleteObject(SelectObject(hdc, hTmp));

}

Файл Resource.h

//{{NO\_DEPENDENCIES}}

// Microsoft Visual C++ generated include file.

// Used by Ticker.rc

//

#define IDC\_MYICON 2

#define IDD\_TICKER\_DIALOG 102

#define IDS\_APP\_TITLE 103

#define IDD\_ABOUTBOX 103

#define IDM\_ABOUT 104

#define IDM\_EXIT 105

#define IDI\_TICKER 107

#define IDI\_SMALL 108

#define IDC\_TICKER 109

#define IDR\_MAINFRAME 128

#define IDB\_PNG1 129

#define IDB\_BITMAP1 131

#define IDI\_ICON1 132

#define ID\_32771 32771

#define ID\_32772 32772

#define ID\_32773 32773

#define ID\_32774 32774

#define ID\_DIGITCLOCK 32775

#define ID\_ARROWCLOCK 32776

#define ID\_ARROWCOLOR 32777

#define ID\_BACKGROUNDCOLOR 32778

#define ID\_CLOCKCOLOR 32779

#define IDC\_STATIC -1

// Next default values for new objects

//

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_NO\_MFC 1

#define \_APS\_NEXT\_RESOURCE\_VALUE 133

#define \_APS\_NEXT\_COMMAND\_VALUE 32780

#define \_APS\_NEXT\_CONTROL\_VALUE 1000

#define \_APS\_NEXT\_SYMED\_VALUE 110

#endif

#endif