Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системное программирование (СП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

ДИСПЕТЧЕР ЗАДАЧ

БГУИР КР 1-40 01 01 217 ПЗ

Студент: гр. 251002 Пунько К. Ю.

Руководитель: Алексеев И. Г.

Минск 2024

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ПОИТ
_____
(подпись)
_____2024 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Пунько Кириллу Юрьевичу

1. Тема работы Диспетчер задач
2. Срок сдачи студентом законченной работы 14.12.2024 г.
3. Исходные данные к работе язык программирования C++
4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Анализ прототипов, литературных источников и моделирование предметной области;

2. Анализ требований к программному средству и разработка функциональных требований;

3. Проектирование программного средства;

4. Создание (конструирование) программного средства;

5. Тестирование, проверка работоспособности и анализ полученных результатов;

6. Руководство по установке и использованию;

Список используемой литературы

Заключение

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. " Диспетчер задач", А1, схема программы, чертеж.

6. Консультант по курсовой работе

Алексеев И. Г.

7. Дата выдачи задания 08.09.2024

8. Календарный график работы над курсовой работой на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 30.09.2024 – 15 % готовности работы;

разделы 2, 3 к 15.10.2024 – 30 % готовности работы;

разделы 4, 5 к 15.11.2024 – 60 % готовности работы;

раздел 6 к 10.12.2024 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 14.12.2024 – 100 % готовности работы.

Защита курсовой работы с 02.12.2024 по 19.12.2024 г.

РУКОВОДИТЕЛЬ Алексеев И. Г.

(подпись)

Задание принял к исполнению 08.09.2024

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одними из важных понятий для любой актуальной в современных реалиях операционной сети являются понятия процессов и потоков. В операционной системе Windows процесс — это экземпляр выполняемой программы, включая текущие значения счетчика команд, регистров и переменных. Поток — это последовательность инструкций, которые выполняются параллельно с другими потоками. Каждая программа создает по меньшей мере один поток: основной, который запускает функцию main(). Программа, использующая только главный поток, является однопоточной; если добавить один или более потоков, она станет многопоточной.

Настоящий центральный процессор постоянно переключается между процессами, но, чтобы понять систему, куда проще думать о наборе процессов, запущенных в параллельно, чем пытаться отслеживать, как центральный процессор переключается между программами. Это постоянное переключение между процессами называется мультипрограммированием, или многозадачным режимом работы.

Для корректной работы с программами в операционных системах необходим Диспетчер задач. Диспетчер задач суть есть утилита, которая предоставляет информацию о процессах и потоках, хранящуюся внутри ОС.

Таким образом, Диспетчер задач определяется как компьютерная программа для вывода на экран списка запущенных процессов, их потоков и потребляемых ими ресурсов.

Цель данной курсовой работы – создать Диспетчер задач, позволяющий анализировать информацию о запущенных процессах и о нагруженности компьютера в данный момент времени.

В ходе выполнения курсовой работы я постараюсь найти решение таким техническим вопросам как:

1. Представление информации о процессах в операционной системе Windows 11;
2. Использование API, работающих с процессами и потоками в ОС Windows.

1 АНАЛИЗ ПРОТОТИПОВ И ПОСТАНОВКА ЗАДАЧИ

1.1 Анализ прототипов

Данная курсовая работа ориентирована на анализ нагруженности ресурсов компьютера. Для того, чтобы иметь представление о функционале Диспетчера задач, рассмотрим уже существующие в качестве прототипов.

1.1.1 Диспетчер задач

Диспетчер задач — компьютерная программа (утилита) для вывода на экран списка запущенных процессов и потребляемых ими ресурсов (в частности статус, процессорное время и потребляемая оперативная память). В качестве дополнительных функций, диспетчер задач может предложить возможность завершить один из процессов или присвоить ему другой приоритет.

Преимущества:

- Даёт весь необходимый функционал для работы с процессами в операционной системе Windows.
- Поставляется вместе с операционной системой и встроен в неё.
- Удобен и для пользователя, который плохо осведомлён о процессах в ОС Windows.

Недостатки:

- Относительно неудобный интерфейс.

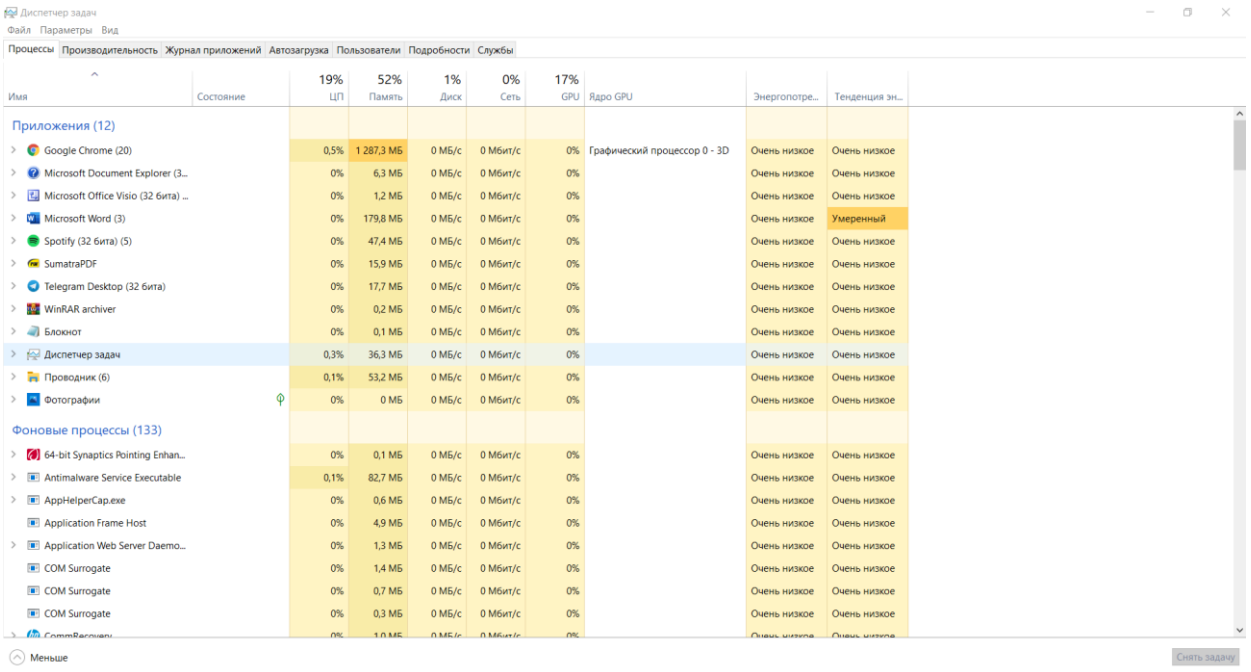

Рисунок 1.5 – интерфейс Диспетчера задач

1.1.2 Process Hacker

Process Hacker – аналог Диспетчера задач, но имеющий несколько более широкий функционал. Например, графики и статистика, предоставляемые данной утилитой, позволяют отследить нехватку ресурсов и неконтролируемые процессы, а в случае, если пользователь не может закрыть какой-то файл, то данная программа позволит отследить процессы, которые этот файл используют

Преимущества:

– Имеет более расширенный, относительно Диспетчера задач, функционал.

– Более удобный интерфейс.

Недостатки:

– Без достаточных знаний о процессах в ОС Windows пользователь не сможет использовать все возможности утилиты.

– Поставляется только на английском языке

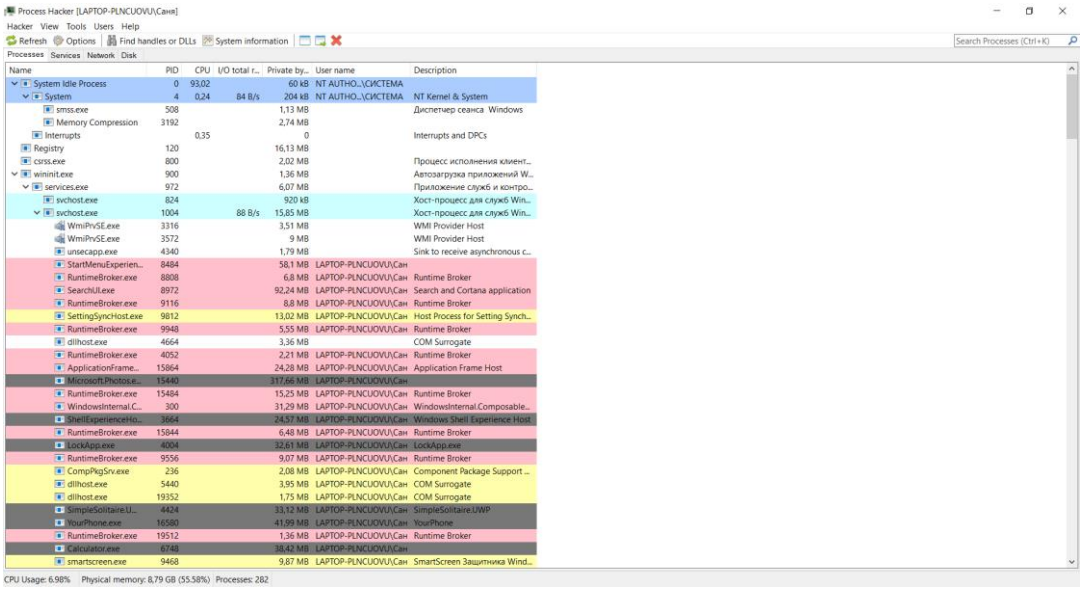

Рисунок 1.6 – интерфейс Process Hacker

1.1.3 Moo0 System Monitor

Moo0 System Monitor – удобный виджет для мониторинга загрузки ресурсов компьютера. Большая часть функций уже описанных выше программ отсутствует, но удобность использования и изначальное позиционирование данного программного продукта перекрывает большинство недостатков.

Преимущества:

– Удобен для мониторинга ресурсов компьютера.

– Гибкий в плане кастомизации графический интерфейс.

Недостатки:

– Не способен предоставить полный функционал своих аналогов.

Рисунок 1.7 – интерфейс виджета Moo0 System Monitor

Вывод: в ходе работы следует создать программное средство, которое способно будет предоставить исчерпывающую информацию о занятых ресурсах компьютера пользователя.

1.2 Постановка задачи

Программное средство должно представлять собой программу для анализа ресурсов компьютера. Программа должна обеспечивать выполнение следующих функций:

– вывод пользователю списка запущенных в ОС процессов, количество потоков, которые использует процесс и номера каждого процесса;

– динамично отображать данные о потребляемых ресурсах компьютера пользователя;

– предоставлять графическую информацию об изменении потребляемых ресурсов компьютера.

Для разработки я выбрал среду Visual Studio 2022 Community Edition. Данная среда является неполной версией Professional-пакета поставляемой для учебных целей студентам и разработчикам open-sourse программного обеспечения. Для работы с графическим интерфейсом было решено использовать средства WinAPI, так как с помощью данной технологии можно и получать информацию о запущенных в операционной системе процессах и их потоках, и создавать графические интерфейсы. В качестве языка программирования выбран язык C++, так как он является оптимальным для работы с технологией WinAPI, если следовать рекомендациям разработчиков данной технологии.

2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Описание функциональных требований

Программное средство для функциональности должно работать напрямую с операционными системами семейства Windows. Должен существовать алгоритм получения информации о запущенных процессах и потребляемых ресурсов персонального компьютера пользователя.

Основными функциями программного средства являются создание списка запущенных процессов, получения информации о потоках и расчёт потребляемых ресурсов ПК.

Для моего программного средства были определены следующие функциональные требования:

1. Наличие графиков потребления ресурсов физической памяти и центрального процессора, которые должны строиться в режиме реального времени;

2. Вывод точных цифр, отображающих размер потребляемых ресурсов ПК (в частности, физической памяти и центрального процессора);

3. При нажатии на кнопку вывода списка процессов, программа должна выводить список запущенных процессов ПК, количество используемых этими процессами потоков и идентификационные номера этих процессов в ОС;

4. Возможность завершить процесс;

2.2 Описание прочих требований

При выполнении работы определяются дополнительные требования:

- Для более удобного анализа нагруженности ресурсов ПК выводимые на экран значения должны обновляться в режиме реального времени.

- На экран должны выводиться значения, которые показывают процентное отношение потребляемых ресурсов ПК относительно доступных для использования.

- Должны выводится графики, которые схематично отображали бы то, что описано пунктом выше.

- Интерфейс программного средства должен быть простым и удобным.

- Программное средство должно работать на устройствах под управлением операционных систем семейства Windows.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка алгоритма получения значений потребляемых ресурсов ПК

3.1.1 Алгоритм получения значений потреблямых ресурсов оперативной памяти ПК

Данный алгоритм представлен функцией, которая, используя функции WinAPI, переносит некоторые значения, связанные с потреблением оперативной памяти, на окно приложения.

Схема алгоритма представлена на рисунке А.1.

3.1.2 Алгоритм получения списка запущенных процессов ОС ПК

После того, как приложение вывело значения потребляемых ресурсов, оно должно, по нажатию на соотвествующую кнопку в меню, находящемуся в верхней части окна приложения, вывести названия каждого запущенного процесса, его ID в ОС и количество используемых этим процессом потоков.

Схема алгоритма представлена на рисунке А.2.

3.2 Разработка алгоритма создания окна в Windows

Для того, чтобы создать окно для приложения на WinAPI, необходимо объявить функцию, которая будет получать сообщения от операционной системы. Функция должна обязательно называться WndProc и принимать четыре параметра:

- Дескриптор окна, указатель на него
- Идентификатор сообщения
- Первый и второй параметры сообщения

В этой функции и будут обрабатываться сообщения от Windows.

Если обработка не будет совершаться, то будет вызываться WinAPI-функция DefWindowProc, которая умеет обрабатывать значения по умолчанию и принимает те же значения, что и WndProc.

Схема алгоритма представлена на рисунке А.3.

3.3 Разработка алгоритма вывода всех необходимых значений на экран

После того, как координаты получены, определяются цвета и радиусы для каждой сферы. Радиусы пропорциональны реальным радиусам Ван-дер-Ваальса, используемым в благородных газах. Цвета определяются также по химическому элементу аналогично тому, как это используется в приложении JMol. Для того, чтобы большая часть изображения была охвачена, вводится коэффициент масштабирования, в начальный момент равный -7. Он представляет собой дополнительное слагаемое в оси аппликат. Он может быть изменен с помощью прокрутки мыши в пропорции: одно деление к 0.2 единицам оси.

4 КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

4.1 Проектирование интерфейса программы

Используемая мною технология WinAPI позволяет создавать нативные для каждой платформы семейства Windows интерфейсы.

Рассмотрим компоненты интерфейса, которые я использовал.

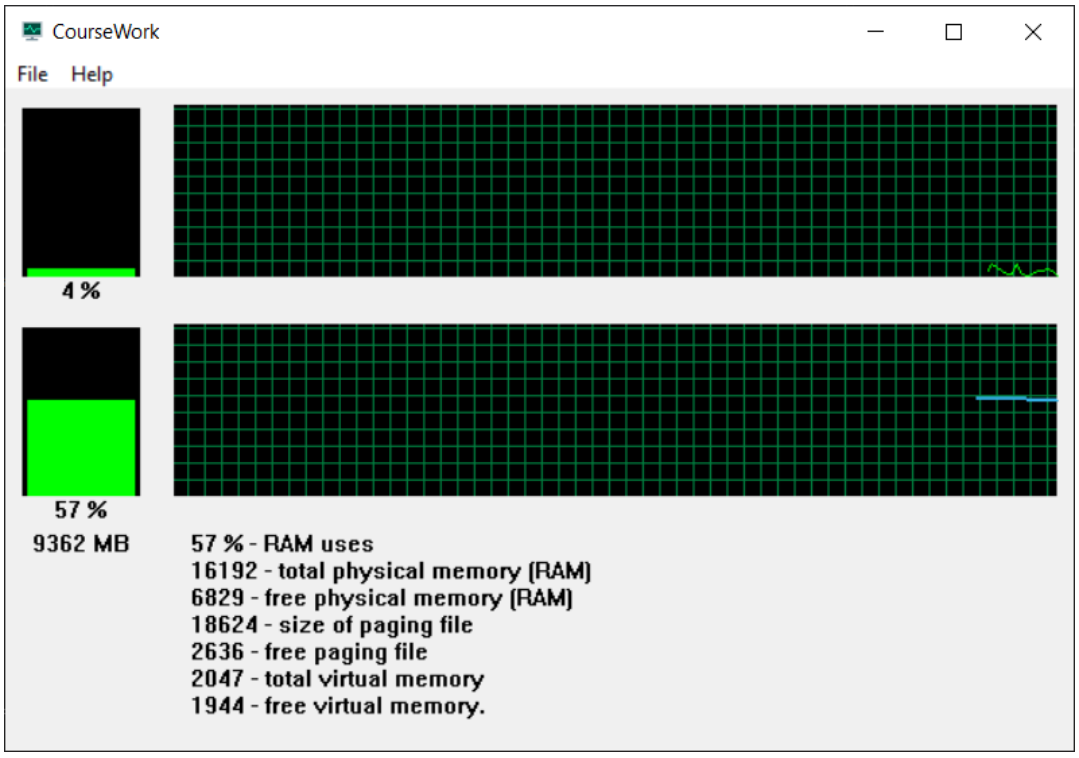

Рисунок 4.1 – интерфейс приложения

Приложение при запуске имеет одно окно, содержащее графики и информацию об использовании памяти. При нажатии на кнопку Processes открывается второе окно со списком процессов, а также с кнопкой Terminate process для завершения процесса.

4.2 Структура модулей программы

Программа состоит из следующих файлов:

1. Файлы модуля приложения CourseWork.h и CourseWork.cpp;

2. Файл ресурсов Resourse.h;

3. Файлы прекомпилируемых заголовков stdafx.h и stdafx.cpp;

4. Файл прекомпилируемого заголовка targetver.h;

4.3 Описание модуля CourseWork

Модуль CourseWork является основным модулем программы, в котором прописаны все основные алгоритмы. Заголовочный файл CourseWork.h нужен для подключения файла ресурсов. В таблице 4.1 представлены все методы данного модуля.

Таблица 4.1 Открытые методы модуля CourseWork

Имя метода	**Описание**	**Заголовок метода**	**Имя параметра**	**Назначение параметра**
_tWinMain	Точка входа прикладной программы	int APIENTRY tWinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance, _In_LPTSTR lpCmdLine, _In_int nCmdShow)	hInstance	Дескриптор текущего экземпляра окна
			hPrevInstance	Дескриптор предыдущего экземпляра окна
			lpCmdLine	Указатель на строку
			nCmdShow	Показывает состояние окна

Продолжение таблицы 4.1

MyRegisterClass	Регистрирует оконный класс	ATOM MyRegisterClass (HINSTANCE hInstance)	hInstance	Дескриптор текущего экземпляра окна
InitInstance	Вызывается каждый раз, когда создаётся новый дескриптор окна	BOOL InitInstance (HINSTANCE hInstance, int nCmdShow)	hInstance	Дескриптор текущего экземпляра окна
			nCmdShow	Состояние показа окна
WndProc	Получение сообщений, отсылаемых операционной системой	LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)	hWnd	Указатель на окно
			message	Сообщение от ОС или пользователя
			wParam	Определяет источник сообщения: элемент управления или акселератор
			lParam	Идентификтор элемента, если это не акселератор
ProcessListWndProc	Обработчик сообщений для окна списка процессов	LRESULT CALLBACK ProcessListWndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)	hWnd	Указатель на окно
			message	Сообщение от ОС или пользователя
			wParam	Определяет источник сообщения: элемент управления или акселератор

Продолжение таблицы 4.1

			lParam	Идентификтор элемента, если это не акселератор
ShowProcessList	Метод для инициализации окна списка процессов	void ShowProcessList(HWND hwndParent)	hwndParent	Указатель на родительское окно
CompareProcesses	Метод для сравнения имен процессов	bool CompareProcesses(const ProcessInfo& a, const ProcessInfo& b)	a	Первый элемент
			b	Второй элемент
TerminateProcessById	Метод для завершения процесса по его ID	bool TerminateProcessById(DWORD processId)	processId	ID процесса, который нужно завершить
CloseSelectedProcess	Метод для завершения выбранного процесса	void CloseSelectedProcess()	-	-
ThreadProc	Метод для получения информации о процессоре и памяти	DWORD WINAPI ThreadProc(LPVOID lpParam)	lpParam	Указатель, предоставляемый юзером

4.4 Описание модуля stdafx

Stdafx является модулем, необходимым для подключения системных заголовочных файлов или для тех заголовочных файлов, которые используются часто, но редко изменяются.

4.5 Описание модуля targetver

Модуль targetver используется для подключения заголовочного файла SDKDDKVer.h, это заголовочный файл, который фактически определяет

необходимые значения, которые представляют каждую версию Windows, IE и т.д.

В результате этапа конструирования создано программное средство. Схема программы представлена в приложении А. Листинг программы представлен в приложении В.

5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

В данном разделе я остановлюсь на тестировании программного средства. В таблицах 5.1 и 5.2 представлены результаты тестирования программного средства.

5.1 Тестирование алгоритма расчёта значений

Таблица 5.1 – тестирование алгоритма расчёта значений

№	Тестируемая функциональ-ность	Последовательность действий	Ожидаемый результат	Полученный результат
1	Корректность значений затраченных ресурсов физической памяти ПК	Запустить приложение, запустить встроенный Диспетчер задач и сравнить значения	Совпадение значений	Тест пройден
2	Корректность значений затраченных ресурсов ЦП ПК	Запустить приложение, запустить встроенный Диспетчер задач и сравнить значения	Совпадение значений	Тест пройден
3	Динамическое изменение значений	Запустить приложение, наблюдать изменение выведенных значений и графиков	Значения изменятся в соответствии со встроенным Диспетчером задач	Тест пройден
4	Изменение показаний при запуске стороннего приложения	Запуск приложения, запуск стороннего приложения и сравнение значений до и после	Увеличение показателей в соответствии со встроенным Диспетчером задач	Тест пройден
5	Изменение показаний при закрытии стороннего приложения	Запуск приложения, закрытие стороннего приложения и сравнение значений до и после	Уменьшение показателей в соответствии со встроенным Диспетчером задач	Тест пройден

Продолжение таблицы 5.1

6	Корректность значений на графике нагруженности ЦП ПК	Сравнение графиков в приложении и во встроенном Диспетчере задач	Совпадение графиков	Тест пройден
7	Корректность значений на графике нагруженности физической памяти ПК	Сравнение графиков в приложении и во встроенном Диспетчере задач	Совпадение графиков	Тест пройден

5.2 Тестирование вывода информации о процессах и потоках

Таблица 5.2 – тестирование вывода информации о процессах и потоках

№	Тестируемая функциональ-ность	Последовательность действий	Ожидаемый результат	Полученный результат
1	Нажатие на кнопки processes	Запуск приложения, нажатие по кнопке processes, открытие окна Process List	Запуск окна с информацией о процессах, их потоках и ID этих процессов в ОС	Тест пройден
2	Изменение количества процессов при запуске стороннего приложения	Запуск приложения, нажатие по кнопке processes, открытие окна Process List, запуск стороннего приложения	Значения изменятся в соответствии со встроенным Диспетчером задач	Тест пройден
3	Корректность вывода количества потоков процесса стороннего приложения	Запуск приложения, нажатие по кнопке processes, открытие окна Process List запуск стороннего приложения	Значения изменятся в соответствии со встроенным Диспетчером задач	Тест пройден
4	Корректность вывода ID процесса стороннего приложения	Запуск приложения, нажатие по кнопке processes, открытие окна Process List запуск стороннего приложения	Значения изменятся в соответствии со встроенным Диспетчером задач	Тест пройден

Продолжение таблицы 5.2

5	Изменение количества процессов после закрытия стороннего приложения	Запуск приложения, нажатие по кнопке processes, открытие окна Process List запуск стороннего приложения, закрытие стороннего приложения	Значения изменятся в соответствии со встроенным Диспетчером задач	Тест пройден
6	Завершение процесса при нажатии по кнопке terminate process	Запуск приложения, нажатие по кнопке processes, открытие окна Process List запуск стороннего приложения, выбор этого приложения из списка процессов, нажатие на кнопку terminate process	Значения изменяются в соответствии со встроенным Диспетчером задач, выбранный процесс закрывается	Тест пройден

Подводя итог, отмечу, что программа отвечает заданным функциональным требованиям, наблюдается стабильность в работе. Вопросов к эстетической части не имеется.

6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

Диспетчер задач обеспечивает вывод корректной информации о доступных и затраченных ресурсов ПК при работе с операционными системами семейства Windows.

Данное программное средство разработано для использования в операционных системах семейства Windows (рис. 6.1).

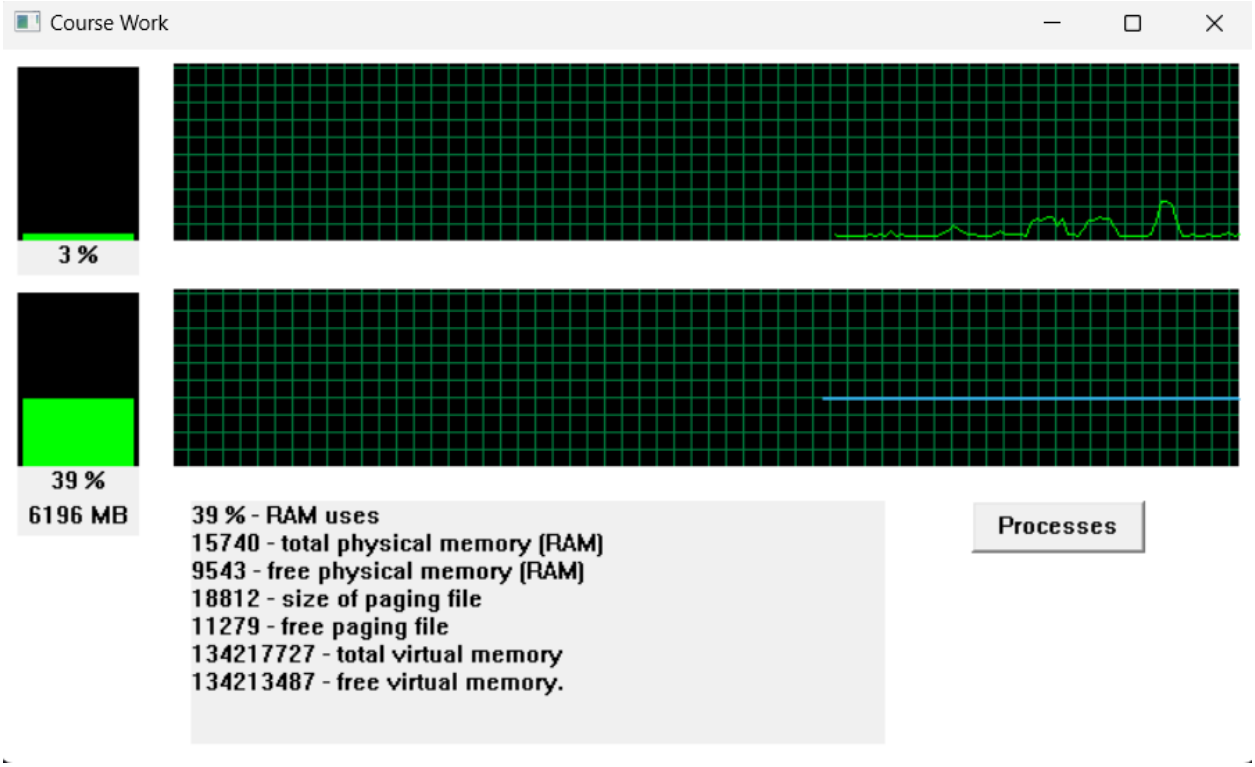

Рисунок 6.1 – Программное средство на устройстве под управлением Windows 10

Для нормальной работы программы папка программного средства должна содержать исполняемый файл CourseWork.exe.

При запуске программы отображается основное окно (рис 6.1).

При запуске программы она сразу же начинает работу с операционной системой и начинает вывод необходимых значений.

Слева видны две диаграммы: выше находится диаграмма, оторбражающая общую нагруженность ЦП, а ниже – общая нагруженность оперативной памяти ПК с точным числом оставшегося пространства (рис 6.2).

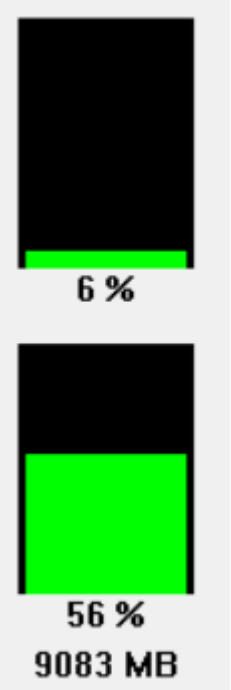

Рисунок 6.2 – диаграммы нагруженности ЦП и оперативной памяти

Правее от них так же соответственно находятся графики, которые отображают нагруженность ЦП и оперативной памяти ПК в конкретный момент времени (рис. 6.3)

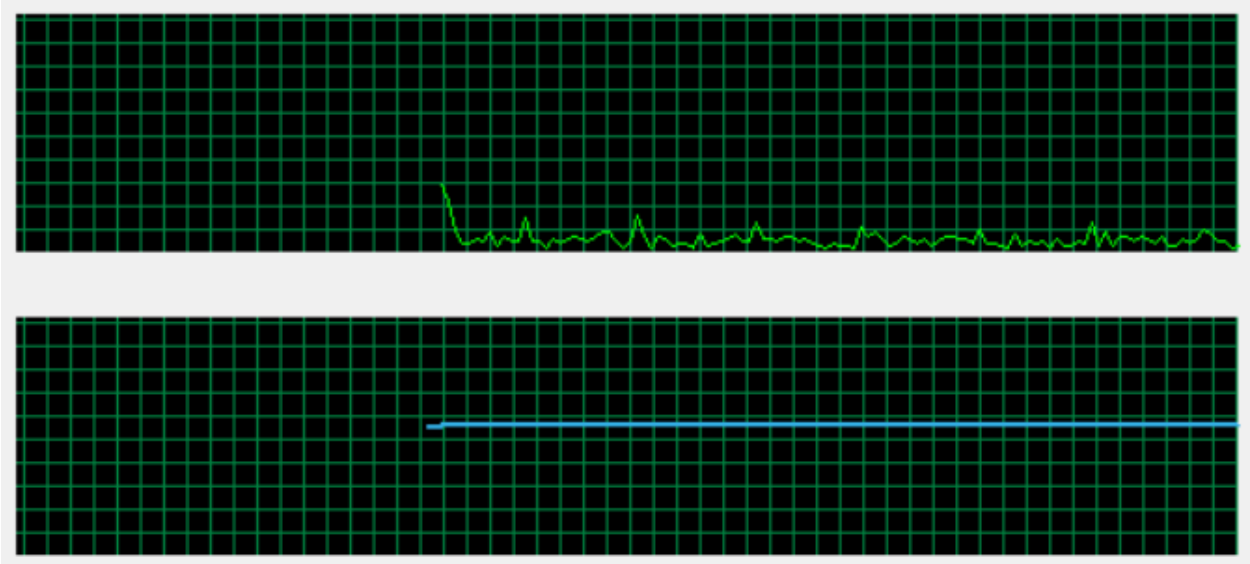

Рисунок 6.3 – графики нагруженности ЦП и оперативной памяти

В самом низу (рис. 6.4) находятся следующие значения:

– Используемая физическая память (в процентах)

– Общее количество физической памяти

– Свободная физическая память

– Размер файла подгрузки

– Незанятая часть файла подгрузки

– Общее количество виртуальной памяти

– Незанятое пространство виртуальной памяти

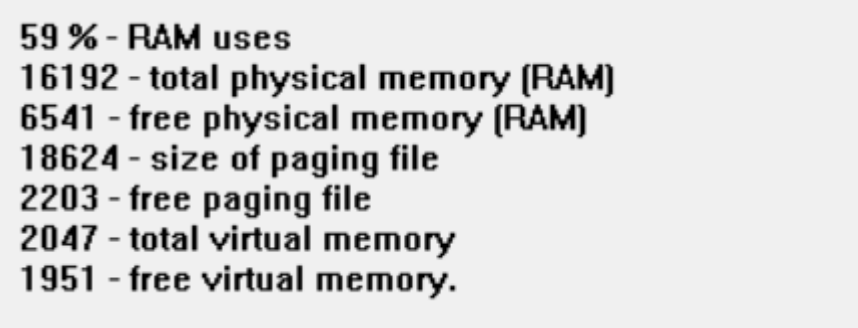

Рисунок 6.4 – значения свободной и занятой памяти разных категорий

По нажатию на кнопку Processes пользователь откроет окно с запущенными в операционной системе процессами, количеством используемыми этими процессами потоков и идентификационные номера этих процессов в операционной системе. Разбиты эти значения на три столбца, вывод происходит справа налево, значения отсортированы по имени процессов (рисунок 6.5).

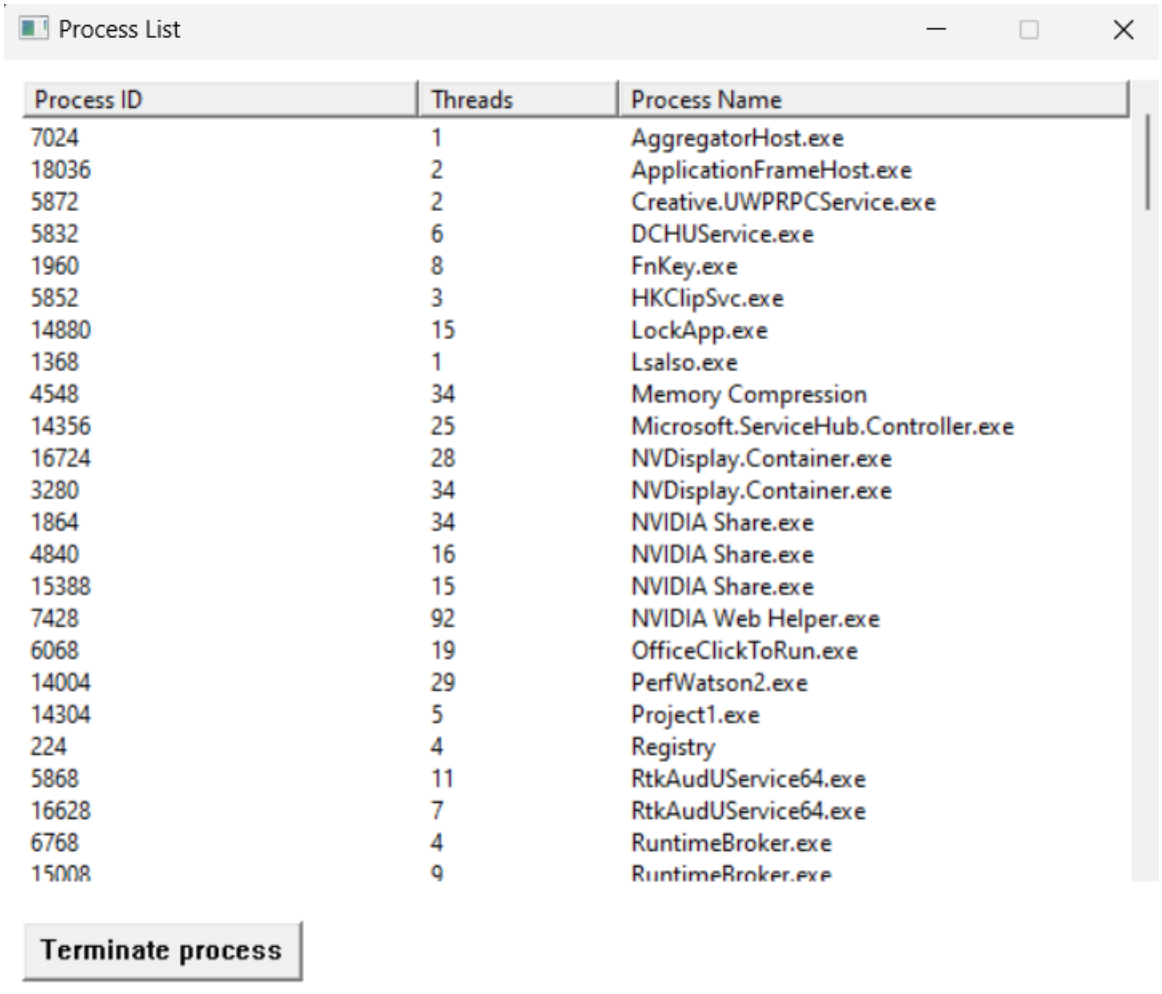

Рисунок 6.5 – окно Processes

В том же окне имеется кнопка Terminate process, которая позволяет завершить процесс.

ЗАКЛЮЧЕНИЕ

В ходе данной работы было создано программное средство «Диспетчер задач», которое предоставляет пользователю информацию о загруженности ресурсов компьютера, список запущенных в операционной системе процессов, количество используемых этими процессами потоков и идентификационные номера процессов в операционной системе.

При этом в ходе работы мною получен опыт работы с процессами и потоками в ОС Windows, а, точнее, с технологией WinAPI. Я использовал встроенные в WinAPI функции, которые позволяют получить информацию о потребляемых ресурсах ПК и о одно- и многопоточных процессах в ОС Windows.

При создании приложения все компоненты интерфейса прописывались вручную, для чего потребовалось хорошо разобраться в англоязычной документации.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

СПИСОК ЛИТЕРАТУРЫ

[1] Таненбаум Э., Бос Х. - Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил. — (Серия «Классика computer science»).

[2] MSDN – Microsoft. – Режим доступа: https://docs.microsoft.com/en-us/welcome-to-docs

[3] Wikipedia [Электронный ресурс] – Поток выполнения – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F

[4] ISO/IEC 14882:2011

ПРИЛОЖЕНИЕ А

Рисунок Б.1 – Схема алгоритма вывода потребляемых ресурсов памяти

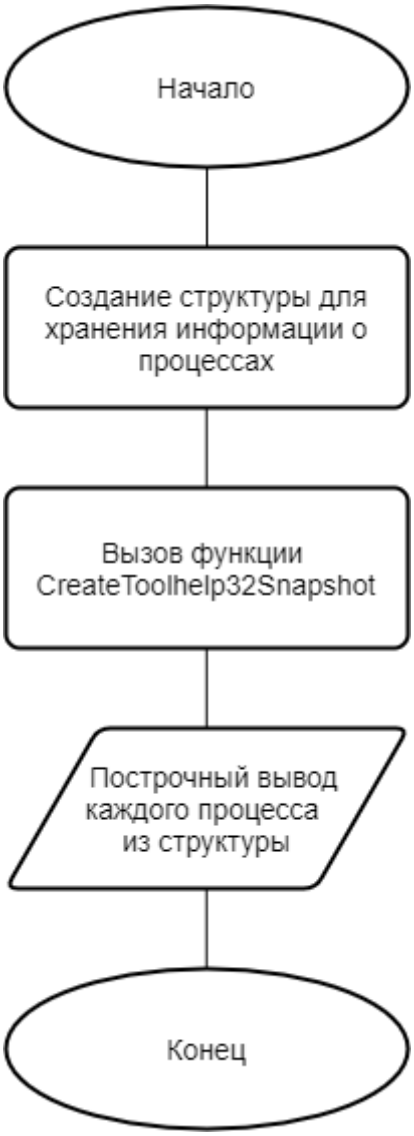

Рисунок Б.2 – Схема алгоритма вывода списка процессов

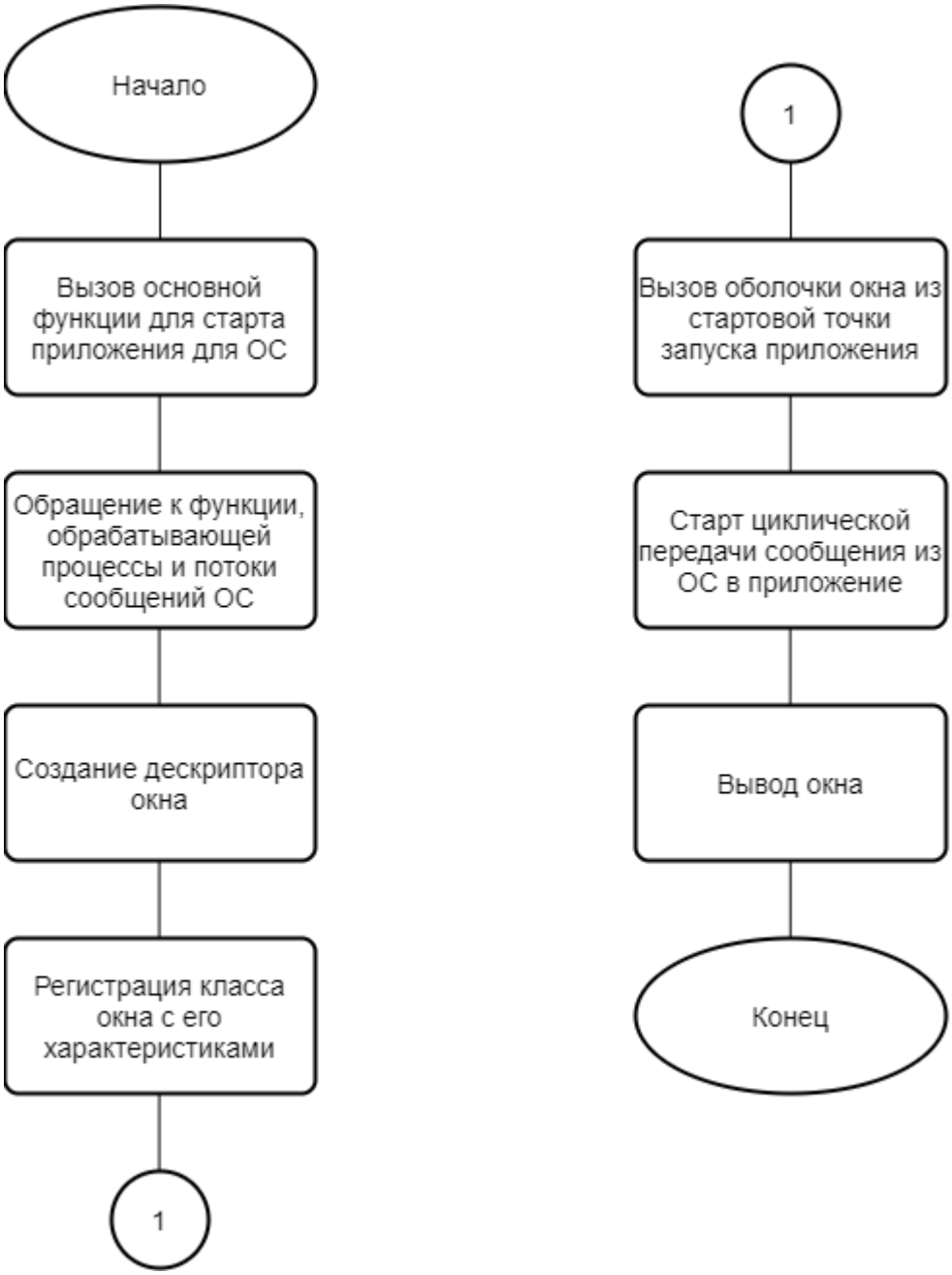

Рисунок Б.3 – Схема алгоритма создания окна в WinAPI

ПРИЛОЖЕНИЕ Б

Листинг файла CourseWork.cpp

```
// CourseWork.cpp : Defines the entry point for the application.
//
#include "stdafx.h"
#include <windows.h>
#include <string>
#include <stdio.h>
#include <commdlg.h>
#include <TlHelp32.h>
#include <stdint.h>
#include<Strsafe.h>
#include <memory>
#include <vector>
#include <algorithm>
#include <commctrl.h>

#pragma comment(lib, "comctl32.lib")

#define MAX_LOADSTRING 100
#define DIV 1048576
#define ID_CPU 2000
#define ID_MEM 2001
#define ID_MEM2 2003
#define ID_EDIT 2002
#define SIZE 400
#define ID_PROCCESSES 106
#define ID_PROCESS_LISTBOX 103

HINSTANCE hInst;
TCHAR szTitle[MAX_LOADSTRING] = L"Course Work";
TCHAR szWindowClass[MAX_LOADSTRING] = L"MyWindowClass";

WCHAR cpu_char[6] = { ' ' };
WCHAR mem_char[8] = { ' ' };
WCHAR buffer[8068] = { ' ' };
WCHAR temp[8068] = { ' ' };
double cpu = -1, memory = -1;
int count = 0, cpu_mas[SIZE] = { -1 }, mem_mas[SIZE] = { -1 };
struct ProcessInfo {
DWORD processID;
int threadCount;
std::wstring processName;
};

FILETIME idleTime;
FILETIME kernelTime;
FILETIME userTime;
FILETIME last_idleTime;
FILETIME last_kernelTime;
FILETIME last_userTime;

static THREADS Thread;

ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

```
DWORD WINAPI ThreadProc(LPVOID);
LRESULT CALLBACK ProcessListWndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam);

HWND hwndProcessList;
HWND hwndListBox;


int APIENTRY _tWinMain(_In_ HINSTANCE hInstance,
_In_opt_ HINSTANCE hPrevInstance,
_In_ LPTSTR    lpCmdLine,
_In_ int       nCmdShow)
{
UNREFERENCED_PARAMETER(hPrevInstance);
UNREFERENCED_PARAMETER(lpCmdLine);

MSG msg;
HACCEL hAccelTable;

MyRegisterClass(hInstance);

if (!InitInstance(hInstance, nCmdShow))
{
       return FALSE;
}

hAccelTable = LoadAccelerators(hInstance, NULL);

while (GetMessage(&msg, NULL, 0, 0))
{
       if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
       {
              TranslateMessage(&msg);
              DispatchMessage(&msg);
       }
}

return (int)msg.wParam;
}

ATOM MyRegisterClass(HINSTANCE hInstance)
{
WNDCLASSEX 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(NULL, IDI_APPLICATION);
wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
wcex.lpszMenuName = NULL;
wcex.lpszClassName = szWindowClass;
wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

return RegisterClassEx(&wcex);
}
// Функция для сортировки по названию процесса
```

```
bool CompareProcesses(const ProcessInfo& a, const ProcessInfo& b) {
return a.processName < b.processName;
}
void ShowProcessList(HWND hwndParent)
{
// Создание нового окна для отображения процессов
hwndProcessList = CreateWindowEx(0, L"STATIC", L"Process List",
WS_OVERLAPPEDWINDOW & ~WS_SIZEBOX & ~WS_MAXIMIZEBOX | WS_VISIBLE,
	200, 200, 600, 510, hwndParent, NULL, GetModuleHandle(NULL), NULL);

// Установка WndProc для окна с процессами
SetWindowLongPtr(hwndProcessList, GWLP_WNDPROC, (LONG_PTR)ProcessListWndProc);

INITCOMMONCONTROLSEX icex;
icex.dwICC = ICC_LISTVIEW_CLASSES;
InitCommonControlsEx(&icex);

// Создание ListBox для отображения процессов
hwndListBox = CreateWindow(WC_LISTVIEW, NULL, WS_CHILD | LVS_REPORT |
LVS_SINGLESEL | LVS_SHOWSELALWAYS | WS_VISIBLE, 10, 10, 570, 400,
hwndProcessList, (HMENU)ID_PROCESS_LISTBOX, GetModuleHandle(NULL), NULL);

ListView_SetExtendedListViewStyle(hwndListBox, LVS_EX_FULLROWSELECT);
if (hwndListBox == NULL) {
	DWORD error = GetLastError();
	wchar_t buffer[256];
	wsprintf(buffer, L"Ошибка при создании окна: %lu", error);
	MessageBox(hwndProcessList, buffer, L"Ошибка", MB_OK);
}
// Добавление колонок
LVCOLUMN lvColumn;
lvColumn.mask = LVCF_TEXT | LVCF_WIDTH | LVCF_SUBITEM;

lvColumn.pszText = const_cast<LPWSTR>(L"Process ID");
lvColumn.cx = 198;
ListView_InsertColumn(hwndListBox, 0, &lvColumn);

lvColumn.pszText = const_cast<LPWSTR>(L"Threads");
lvColumn.cx = 100;
ListView_InsertColumn(hwndListBox, 1, &lvColumn);

lvColumn.pszText = const_cast<LPWSTR>(L"Process Name");
lvColumn.cx = 555-300;
ListView_InsertColumn(hwndListBox, 2, &lvColumn);

// Кнопка "Закрыть"
HWND hwndCloseButton = CreateWindowEx(0, L"BUTTON", L"Terminate process",
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 10, 430, 140, 30, hwndProcessList,
(HMENU)IDCANCEL, GetModuleHandle(NULL), NULL);

HANDLE processHandle;
PROCESSENTRY32 processes;
WCHAR buffer[255];
std::vector<ProcessInfo> processList;

// Создание снапшота процессов
processHandle = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
processes.dwSize = sizeof(PROCESSENTRY32);
if (Process32First(processHandle, &processes))
{
	do
```

```
	{
		// Добавляем информацию о процессе в список
		ProcessInfo pInfo;
		pInfo.processID = processes.th32ProcessID;
		pInfo.threadCount = processes.cntThreads;
		pInfo.processName = processes.szExeFile;

		processList.push_back(pInfo);
	} while (Process32Next(processHandle, &processes));
}
CloseHandle(processHandle);
// Сортировка процессов по имени
std::sort(processList.begin(), processList.end(), CompareProcesses);

// Очищаем ListView перед добавлением новых данных
ListView_DeleteAllItems(hwndListBox);

// Добавляем данные в ListView
for (size_t i = 0; i < processList.size(); ++i) {
	LVITEM lvItem;
	lvItem.mask = LVIF_TEXT;
	lvItem.iItem = i;

	// Добавляем процесс ID в первую колонку
	lvItem.iSubItem = 0;
	swprintf_s(buffer, L"%d", processList[i].processID);
	lvItem.pszText = buffer;
	ListView_InsertItem(hwndListBox, &lvItem);

	// Добавляем количество потоков во вторую колонку
	lvItem.iSubItem = 1;
	swprintf_s(buffer, L"%d", processList[i].threadCount);
	lvItem.pszText = buffer;
	ListView_SetItem(hwndListBox, &lvItem);

	// Добавляем имя процесса в третью колонку
	lvItem.iSubItem = 2;
	lvItem.pszText = const_cast<LPWSTR>(processList[i].processName.c_str());
	ListView_SetItem(hwndListBox, &lvItem);
}
}
bool TerminateProcessById(DWORD processId) {
// Открываем процесс с правом завершения
HANDLE hProcess = OpenProcess(PROCESS_TERMINATE, FALSE, processId);
if (hProcess == NULL) {
	return false;
}

// Завершаем процесс
BOOL result = TerminateProcess(hProcess, 0);
if (result == 0) {
	CloseHandle(hProcess);
	return false;
}

// Закрываем дескриптор процесса
CloseHandle(hProcess);
return true;
}
void CloseSelectedProcess() {
int selectedIndex = (int)SendMessage(hwndListBox, LVM_GETNEXTITEM, -1,
```

```
LVNI_SELECTED);
if (selectedIndex == -1) {
	MessageBox(hwndListBox, L"Не выбран процесс.", L"Ошибка", MB_OK |
MB_ICONERROR);
	return;
}

// Извлекаем текст из первой колонки (где хранится ID процесса)
WCHAR buffer[255];
LVITEM lvItem = {};
lvItem.iItem = selectedIndex;
lvItem.iSubItem = 0; // Первая колонка
lvItem.mask = LVIF_TEXT;
lvItem.pszText = buffer;
lvItem.cchTextMax = sizeof(buffer) / sizeof(buffer[0]);

if (!SendMessage(hwndListBox, LVM_GETITEMTEXT, (WPARAM)selectedIndex,
(LPARAM)&lvItem)) {
	MessageBox(hwndListBox, L"Не удалось получить данные строки.", L"Ошибка",
	MB_OK | MB_ICONERROR);
	return;
}

// Преобразуем ID процесса из текста
DWORD processId = 0;
if (swscanf_s(buffer, L"%lu", &processId) != 1 || processId == 0) {
	MessageBox(hwndListBox, L"Не удалось извлечь ID процесса.", L"Ошибка",
	MB_OK | MB_ICONERROR);
	return;
}

// Завершаем процесс
if (TerminateProcessById(processId)) {
	MessageBox(hwndListBox, L"Процесс успешно завершен.", L"Успех", MB_OK |
	MB_ICONINFORMATION);

	// Удаляем строку из ListView
	SendMessage(hwndListBox, LVM_DELETEITEM, (WPARAM)selectedIndex, 0);
}
else {
	MessageBox(hwndListBox, L"Не удалось завершить процесс.", L"Ошибка",
	MB_OK | MB_ICONERROR);
}
}

LRESULT CALLBACK ProcessListWndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
switch (msg)
{
case WM_COMMAND:
	if (LOWORD(wParam) == IDCANCEL)  // Если нажата кнопка "Закрыть"
	{
		CloseSelectedProcess();
	}
	break;

}

return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
HWND hWnd;

hInst = hInstance;

hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW & ~WS_SIZEBOX,
       300, 200, 740, 450, NULL, NULL, hInstance, NULL);

if (!hWnd)
{
       return FALSE;
}

HWND edit_cpu = CreateWindow(L"static", L"CPU:", WS_CHILD | WS_VISIBLE |
ES_CENTER, 10, 110, 70, 20, hWnd, (HMENU)ID_CPU, hInstance, NULL);

HWND edit_mem = CreateWindow(L"static", L"Memory:", WS_CHILD | WS_VISIBLE |
ES_CENTER, 10, 240, 70, 20, hWnd, (HMENU)ID_MEM, hInstance, NULL);

HWND edit_mem2 = CreateWindow(L"static", L"", WS_CHILD | WS_VISIBLE | ES_CENTER,
       10, 260, 70, 20, hWnd, (HMENU)ID_MEM2, hInstance, NULL);

HWND edit2 = CreateWindow(L"static", L"", WS_CHILD | WS_VISIBLE,
       110, 260, 400, 140, hWnd, (HMENU)ID_EDIT, hInstance, NULL);

HWND btnShowProccess = CreateWindow(L"button", L"Processes", WS_CHILD |
WS_VISIBLE | BS_PUSHBUTTON,
       560, 260, 100, 30, hWnd, (HMENU)ID_PROCCESSES, hInstance, NULL);

Thread.handleThread = NULL;
Thread.ThreadFucntion = ThreadProc;
Thread.handleDialog = hWnd;
Thread.time = 0;
Thread.handleThread = CreateThread(NULL, 0, Thread.ThreadFucntion, NULL,
       NULL, &Thread.threadId);

for (int i = SIZE - 1; i >= 0; i--) {
       cpu_mas[i] = -1;
       mem_mas[i] = -1;
}

ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);

return TRUE;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
int wmId, wmEvent;
PAINTSTRUCT ps;
HDC hdc;

HPEN hpen, hpen_cpu_graph, hpen_mem_graph;
RECT rect, rect_cpu, rect_cpu_graph, rect_mem, rect_mem_update,
       rect_mem_graph, rect_window;
POINT Min;
MINMAXINFO* pInfo;
```

```
switch (message)
{
case WM_GETMINMAXINFO: //Getting message from Windows
	pInfo = (MINMAXINFO*)lParam;
	Min = { 650, 450 };
	pInfo->ptMinTrackSize = Min;
	return 0;

case WM_COMMAND:
	wmId = LOWORD(wParam);
	wmEvent = HIWORD(wParam);
	switch (wmId)
	{
	case 105:  // Обработка выхода
		DestroyWindow(hWnd);
		break;
	case ID_PROCCESSES: {
		ShowProcessList(hWnd);
		break;
	}
	default:
		return DefWindowProc(hWnd, message, wParam, lParam);
	}
	break;
case WM_PAINT:
	GetClientRect(hWnd, &rect_window);
	hdc = BeginPaint(hWnd, &ps);

	SetRect(&rect, 10, 10, 80, 110);
	if (cpu_mas[0] >= 0)
	{
		SetRect(&rect_cpu, 13, 110 - cpu, 77, 110);
	}

	SetRect(&rect_cpu_graph, 100, 8, rect_window.right - 10, 110);
	SetRect(&rect_mem, 10, 140, 80, 240);
	SetRect(&rect_mem_update, 13, rect_mem.bottom - memory, 77,
		rect_mem.bottom);
	SetRect(&rect_mem_graph, 100, rect_mem.top - 2, rect_window.right - 10,
		rect_mem.bottom);

	FillRect(hdc, &rect, CreateSolidBrush(RGB(0, 0, 0)));
	FillRect(hdc, &rect_cpu, CreateSolidBrush(RGB(0, 255, 0)));
	FillRect(hdc, &rect_cpu_graph, CreateSolidBrush(RGB(0, 0, 0)));
	FillRect(hdc, &rect_mem, CreateSolidBrush(RGB(0, 0, 0)));
	FillRect(hdc, &rect_mem_update, CreateSolidBrush(RGB(0, 255, 0)));
	FillRect(hdc, &rect_mem_graph, CreateSolidBrush(RGB(0, 0, 0)));

	hpen = CreatePen(PS_SOLID, 0, RGB(0, 128, 64));
	SelectObject(hdc, hpen);

	//grid
	count += 3;

	for (int i = rect_window.right - 10; i > rect_cpu_graph.left; i -= 10)
	{
		if (i - count % 10 >= rect_cpu_graph.left)
		{
			MoveToEx(hdc, i - count % 10, 8, 0);
			LineTo(hdc, i - count % 10, 110);
```

```
                        MoveToEx(hdc, i - count % 10, rect_mem.top - 2, 0);
                        LineTo(hdc, i - count % 10, rect_mem.bottom);
                }

        }

        for (int i = 10; i < 110; i += 10)
        {
                MoveToEx(hdc, rect_cpu_graph.left, i, 0);
                LineTo(hdc, rect_window.right - 10, i);
                MoveToEx(hdc, rect_cpu_graph.left, i + 130, 0);
                LineTo(hdc, rect_window.right - 10, i + 130);
        }

        DeleteObject(hpen);

        //cpu_graph
        hpen_cpu_graph = CreatePen(PS_SOLID, 1, RGB(0, 255, 0));
        SelectObject(hdc, hpen_cpu_graph);

        for (int i = 1, j = 10; i < SIZE; i++, j += 3)
        {
                if ((cpu_mas[i + 1] >= 0) &&
                        ((rect_window.right - j) > rect_cpu_graph.left))
                {
                        MoveToEx(hdc, rect_window.right - j, 110 - cpu_mas[i], 0);
                        LineTo(hdc, rect_window.right - j - 3,
                                110 - cpu_mas[i + 1]);

                }
        }

        DeleteObject(hpen_cpu_graph);

        //memory_graph
        hpen_mem_graph = CreatePen(PS_SOLID, 2, RGB(50, 172, 225));
        SelectObject(hdc, hpen_mem_graph);

        for (int i = 0, j = 10; i < SIZE - 1; i++, j += 3)
        {
                if ((mem_mas[i + 1] >= 0) &&
                        ((rect_window.right - j) > rect_mem_graph.left))
                {
                        MoveToEx(hdc, rect_window.right - j,
                                rect_mem_graph.bottom - mem_mas[i], 0);
                        LineTo(hdc, rect_window.right - j - 3,
                                rect_mem_graph.bottom - mem_mas[i + 1]);
                }
        }

        DeleteObject(hpen_mem_graph);

        EndPaint(hWnd, &ps);
        break;

case WM_DESTROY:
        PostQuitMessage(0);
        break;

default:
```

```
	return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

// Message handler for process list box.
INT_PTR CALLBACK ProcessList(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam)
{
UNREFERENCED_PARAMETER(lParam);
switch (message)
{
case WM_INITDIALOG: {
	HWND hList = GetDlgItem(hDlg, 1000);
	HANDLE processHandle;
	PROCESSENTRY32 processes;
	WCHAR buffer[255];

	// Создание снапшота процессов
	processHandle = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
	processes.dwSize = sizeof(PROCESSENTRY32);
	if (Process32First(processHandle, &processes))
	{
		do
		{
			// Формирование строки для отображения
			wsprintfW(buffer, L" ID [%05d] Threads [%03d] %30s",
				processes.th32ProcessID, processes.cntThreads,
				processes.szExeFile);

			// Добавление строки в список
			SendMessageW(hList, LB_ADDSTRING, 0, (WPARAM)buffer);
		} while (Process32Next(processHandle, &processes));
	}
	CloseHandle(processHandle);

	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;
}
ULONGLONG SubtractTimes(const FILETIME& later, const FILETIME& earlier) {
ULARGE_INTEGER a, b;
a.LowPart = later.dwLowDateTime;
a.HighPart = later.dwHighDateTime;

b.LowPart = earlier.dwLowDateTime;
b.HighPart = earlier.dwHighDateTime;

return a.QuadPart - b.QuadPart;
}
DWORD WINAPI ThreadProc(LPVOID lpParam)
{
THREADS* ti = (THREADS*)lpParam;
```

```
GetSystemTimes(&last_idleTime, &last_kernelTime, &last_userTime);
for (;;)
{
	if (GetSystemTimes(&idleTime, &kernelTime, &userTime) != 0)
	{

		double usr = userTime.dwLowDateTime - last_userTime.dwLowDateTime;
		double ker = kernelTime.dwLowDateTime –
		last_kernelTime.dwLowDateTime;
		double idl = (idleTime.dwLowDateTime –
		last_idleTime.dwLowDateTime);

		double sys = ker + usr;

		last_idleTime = idleTime;
		last_userTime = userTime;
		last_kernelTime = kernelTime;

		if (sys != 0) {
			cpu = (sys - idl) / sys * 100;

			for (int i = SIZE - 1; i > 0; i--) {
				cpu_mas[i] = cpu_mas[i - 1];
			}
			cpu_mas[0] = cpu;
		}


		if (cpu >= 0)
		{
			wsprintfW(cpu_char, L"%d %%", (int)cpu);
			SetDlgItemText(Thread.handleDialog, 2000, cpu_char);
		}
	}

	// Получение состояния памяти
	MEMORYSTATUSEX statex;
	statex.dwLength = sizeof(statex);
	GlobalMemoryStatusEx(&statex);

	// Обновление памяти в UI
	wsprintfW(buffer, L"%ld %% - RAM uses\r\n", statex.dwMemoryLoad);
	memory = statex.dwMemoryLoad;
	for (int i = SIZE - 1; i > 0; i--) {
		mem_mas[i] = mem_mas[i - 1];
	}
	mem_mas[0] = memory;

	wsprintfW(mem_char, L"%d %%", (int)memory);
	SetDlgItemText(Thread.handleDialog, 2001, mem_char);

	wsprintfW(temp, L"%ld - total physical memory (RAM)\r\n",
	 statex.ullTotalPhys / DIV);
	StringCchCat(buffer, sizeof(temp), temp);

	wsprintfW(temp, L"%ld - free physical memory (RAM)\r\n",
	 statex.ullAvailPhys / DIV);
	StringCchCat(buffer, sizeof(temp), temp);

	wsprintfW(mem_char, L"%d MB", (statex.ullTotalPhys - statex.ullAvailPhys)
	 / DIV);
```

```
        SetDlgItemText(Thread.handleDialog, 2003, mem_char);

        wsprintfW(temp, L"%ld - size of paging file\r\n", statex.ullTotalPageFile
        / DIV);
        StringCchCat(buffer, sizeof(temp), temp);
        wsprintfW(temp, L"%ld - free paging file\r\n", statex.ullAvailPageFile /
        DIV);
        StringCchCat(buffer, sizeof(temp), temp);
        wsprintfW(temp, L"%ld - total virtual memory\r\n", statex.ullTotalVirtual
        / DIV);
        StringCchCat(buffer, sizeof(temp), temp);
        wsprintfW(temp, L"%ld - free virtual memory.\r\n", statex.ullAvailVirtual
        / DIV);
        StringCchCat(buffer, sizeof(temp), temp);

        SetDlgItemText(Thread.handleDialog, 2002, buffer);

        InvalidateRect(Thread.handleDialog, NULL, TRUE);
        Sleep(500);
}

}
```

Листинг файла CourseWork.h

```
#pragma once

#include "resource.h"
```

Листинг файла stdafx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
// CourseWork.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

Листинг файла stdafx.cpp

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once

#include "targetver.h"

#define WIN32_LEAN_AND_MEAN
// Exclude rarely-used stuff from Windows headers
// Windows Header Files:
#include <windows.h>

// C RunTime Header Files
#include <stdlib.h>
```

```
#include <malloc.h>
#include <memory.h>
#include <tchar.h>

typedef struct THREADS
{
HANDLE handleThread;
LPTHREAD_START_ROUTINE ThreadFucntion;
HWND handleProgressBar;
HWND handleTimerEdit;
DWORD threadId;
HWND handleDialog;
UINT idcTimeEdit;
UINT idcTimer;
UINT time;
};
```

Листинг файла targetver.h

```
#pragma once

// Including SDKDDKVer.h defines the highest available Windows platform.

// If you wish to build your application for a previous Windows platform, include
//WinSDKVer.h and
// set the _WIN32_WINNT macro to the platform you wish to support before
//including SDKDDKVer.h.

#include <SDKDDKVer.h>
```

ВЕДОМОСТЬ

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КР 1–40 01 01 217 ПЗ	Пояснительная записка	42 с.
	Графические документы	
ГУИР 251002 217 СП	Схема программы «Диспетчер задач»	Формат А1

					БГУИР КР 1-40 01 01 217 Д1		
Изм.	Л.	№ докум.	Подп.	Дата		Лист	Листов
Разраб.		Пунько К.Ю.		14.12.20	Диспетчер задач Ведомость курсовой работы	Т	43 / 43
Пров.		Алексеев И.Г.		14.12.20		Кафедра ПОИТ гр. 251002	