Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

ДИСПЕТЧЕР ЗАДАЧ

БГУИР КР 1-40 01 01 222 ПЗ

Студент: гр. 951002 Мешок А. Д.

Руководитель: асс. Шостак Е. В.

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2020 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Мешку Александру Дмитриевичу

1. Тема работы Диспетчер задач

2. Срок сдачи студентом законченной работы––21.12.2020 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

Шостак Е. В.

7. Дата выдачи задания 08.09.2020

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

раздел 1 к 15.09.2020 – 15 % готовности работы;

разделы 2, 3 к 15.10.2020 – 30 % готовности работы;

разделы 4, 5 к 15.11.2020 – 60 % готовности работы;

раздел 6 к 21.12.2020 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 22.12.2020 – 100 % готовности работы.

Защита курсовой работы с 13.12.2020 по 22.12.2020 г.––––––––––––––––––––

РУКОВОДИТЕЛЬ–––––– Шостак Е. В.

(подпись)

Задание принял к исполнению 08.09.2020–––\_\_\_\_\_\_––

(дата и подпись студента)

Содержание

[Содержание 4](#_Toc59521562)

[Введение 6](#_Toc59521563)

[1 Анализ прототипов, литературных источников и моделирование предметной области 8](#_Toc59521564)

[1.1 Процессы и их состояния 8](#_Toc59521565)

[1.2 Многозадачный режим или Мультипрограммирование 9](#_Toc59521566)

[1.3 Анализ прототипов 12](#_Toc59521567)

[1.4 Постановка задачи 15](#_Toc59521568)

[2 Анализ требований к программному средству и разработка функциональных требований 16](#_Toc59521569)

[2.1 Описание функциональных требований 16](#_Toc59521570)

[2.2 Описание прочих требований 17](#_Toc59521571)

[3 Проектирование программного средства 18](#_Toc59521572)

[3.1 Разработка алгоритма получения значений потребляемых ресурсов ПК 18](#_Toc59521573)

[3.2 Разработка алгоритма создания окна в Windows 18](#_Toc59521574)

[3.3 Разработка алгоритма вывода всех необходимых значений на экран 19](#_Toc59521575)

[4 Конструирование программного средства 19](#_Toc59521576)

[4.1 Проектирование интерфейса программы 19](#_Toc59521577)

[4.2 Структура модулей программы 20](#_Toc59521578)

[4.3 Описание модуля CourseWork 20](#_Toc59521579)

[4.4 Описание модуля Resourse 22](#_Toc59521580)

[4.5 Описание модуля stdafx 22](#_Toc59521581)

[4.6 Описание модуля targetver 22](#_Toc59521582)

[4.9 Описание модуля с иконками для приложения 22](#_Toc59521583)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 23](#_Toc59521584)

[5.1 Тестирование алгоритма расчёта значений 23](#_Toc59521585)

[5.2 Тестирование вывода информации о процессах и потоках 24](#_Toc59521586)

[6 Руководство по установке и использованию 26](#_Toc59521587)

[Заключение 30](#_Toc59521588)

[Список литературы 31](#_Toc59521589)

[Приложение A 32](#_Toc59521590)

[Приложение Б 33](#_Toc59521591)

[Приложение В 36](#_Toc59521592)

[Листинг файла CourseWork.cpp 36](#_Toc59521593)

[Листинг файла CourseWork.h 43](#_Toc59521594)

[Листинг файла Resourse.h 43](#_Toc59521595)

[Листинг файла stdafx.cpp 44](#_Toc59521596)

[Листинг файла stdafx.cpp 44](#_Toc59521597)

[Листинг файла targetver.h 44](#_Toc59521598)

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 10;
2. Использование API, работающих с процессами и потоками в ОС Windows.

В этой пояснительной записке отображены следующие этапы написания курсовой работы:

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.
7. Анализ прототипов, литературных источников и моделирование предметной области
   1. Процессы и их состояния

Как уже было определено выше, процесс — это экземпляр выполняемой программы, включая текущие значения счетчика команд, регистров и переменных.

У процесса есть несколько состояний, которыми можно охарактеризовать его работу в конкретный момент времени. Для объяснения концепции состояний процесса воспользуемся простешим графическим представленем оной, приведенной на рисунке 1.1:

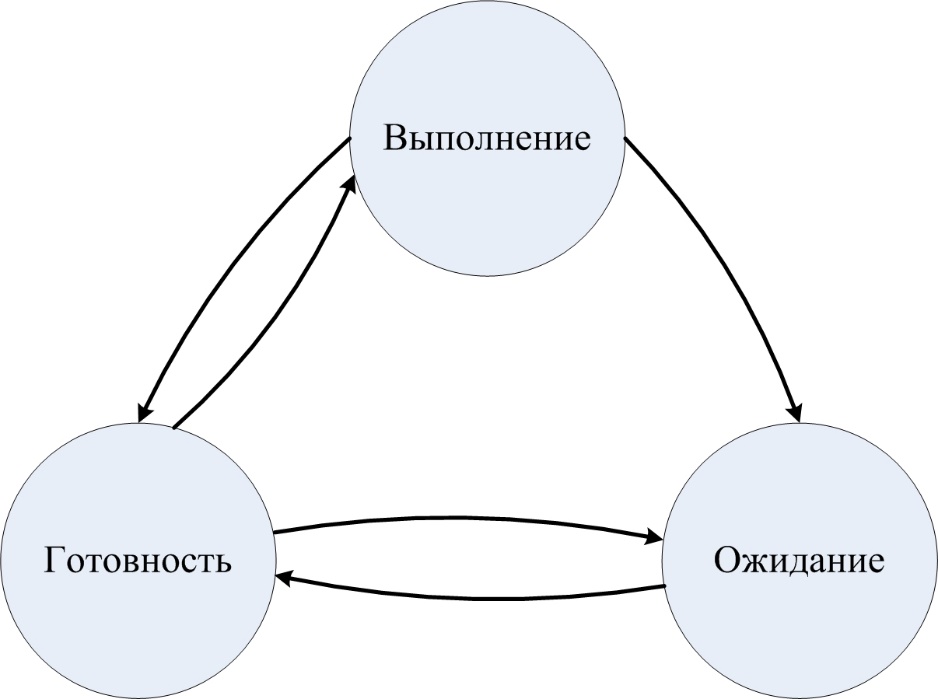


Рисунок 1.1 – графическое представление концепии состояний процесса

Состояние выполнения – активное состояние, при котором процесс непосредственно выполняется центральным процессом. В состоянии выполнения ресурсы компьютера могут

Состояние ожидания – пассивное состояние, во время которого процесс заблокирован и не может быть выполнен, так как процесс ожидает какого-либо события (например, ввод данных).

Состояние готовности также является пассивным, при котором процесс заблокирован, но в конкретном случае данное состояние связано с внешними, не зависящими от самого процесса причинам.

В реальности же состояний процессов на деле куда больше, но простейшая модель, приведённая выше, способна дать исчерпывающее представление о работе процесса в операционной системе.

* 1. Многозадачный режим или Мультипрограммирование
     1. Описывающие процессы информационные структуры

Описать процессы можно с помощью двух информационных структур: контекста процесса и дескриптора процесса.

Контекст процесса содержит информацию о внутреннем состоянии процесса, а также отражает состояние аппаратуры в момент прерывания процесса и включает параметры операционной среды. Содержит часть информации необходимой для возобновления выполнения процесса с прерванного места.

Дескриптор процесса – это информационная структура, которая описывает внешнюю структуру (информацию) процессе (нужна планировщику для выполнения процесса, а также нужна ядру в течение всего жизненного цикла процесса).

В дескриптор процесса входит:

* Идентификатор процесса – уникальный номер процесса в многозадачной операционной системе.
* Состояние процесса.
* Информация о привилегированности процесса – информация о том, разрешается ли процессу выполнять операции, связанные с защитой ОС.
* Информация о расположении кодового сегмента.

Таким образом, при разработке программного обеспечения, мы будем опираться на дескрипторы процессов, запущенных в ОС.

* + 1. Мультипрограммирование на физическом уровне

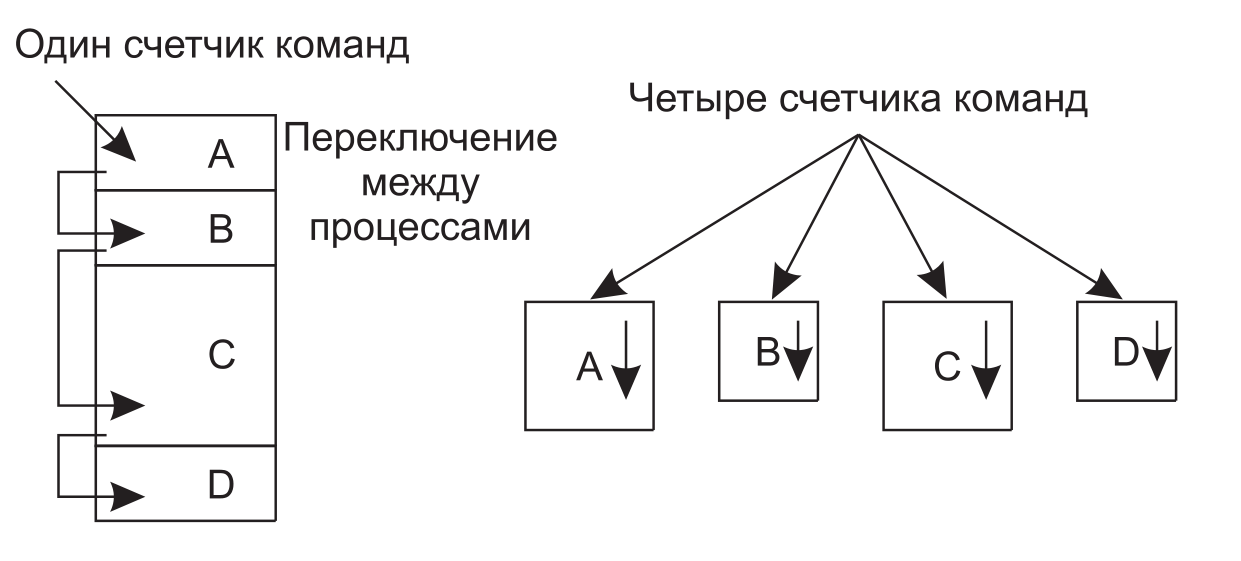


Рисунок 1.2 – представление работы компьютера, работающего с четырьмя программами в многозадачном режиме

На рисунке 1.2 схематично показан компьютер, который работает в многозадачном режиме и имеет в памяти четыре программы, каждая из которых имеет собственный алгоритм управления (то есть свой логический счётчик команд). На самом же деле имеется только один физический счётчик команд, поэтому при запуске каждого процесса его логический счетчик команд загружается в реальный счетчик. Когда работа с процессом будет на некоторое время прекращена, значение физического счетчика команд сохраняется в логическом счетчике команд, размещаемом процессом в памяти.

На рисунке 1.3, в показано, что за довольно длительный период наблюдения продвинулись вперед все процессы, но в каждый отдельно взятый момент времени реально работает только один процесс.

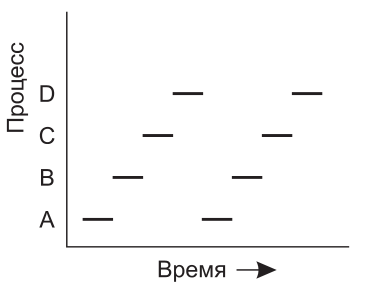


Рисунок 1.3 – представление времени работы процессов с рисунка 1.2

* + 1. Работа с процессом в ОС Windows

Для создания процесса в Windows используется функция CreateProcess.

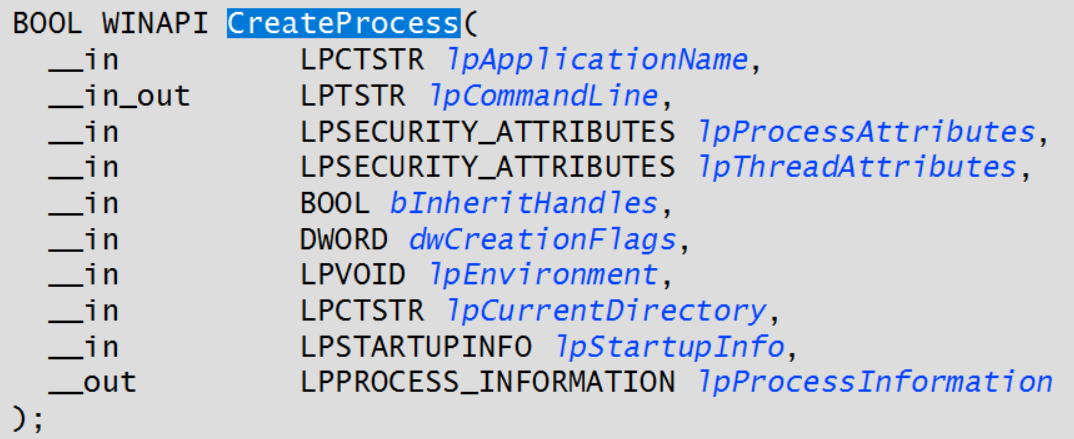


Рисунок 1.4 – функция WinAPI CreateProcess

Функция CreateProcess создаёт процесс и его первый поток. Новый процесс запускается в контексте безопасности вызывающего процесса. Если запуск процесса удался – функция вернёт ненулевое значение. Если запуск процесса провалился – вернёт ноль.

Рассмотрим передаваемые в данную функцию параметры:

* lpApplicationName – указатель на Си-строку, содержащую имя выполняемого модуля. Если данный параметр равен NULL, то имя модуля должно хранится в lpCommandLine первым элементом.
* lpCommandLine – коммандная строка. Здесь передаются параметры запускаемого процесса (путь или имя модуля). Может быть NULL.
* lpProcessAttributes – аттрибуты защиты для нового приложения. Если равен NULL, то ОС задаст параметр по умолчанию.
* lpThreadAttributes – аттрибуты защиты для первого потока запускаемого приложения. Если равен NULL, то ОС задаст параметр по умолчанию.
* bInheritHandles – флаг наследования от производящего запуск процесса (т.е. тут наследуются дескрипторы, причём с такими же параметрами доступа, как и у наследуемого процесса).
* dwCreationFlags – флаг способа задания процесса и его приоритета.
* lpEnviroment – указывает на структуру данных операционной системы семества. Если NULL, то будет использована структура данных родительского процесса.
* lpCurrentDirectory – указывает на текущий диск и/или каталог.
* lpStartupInfo – настройка свойств процесса (например, расположение окон и заголовков)
* lpProcessInformation – информация о процессе. Заполняется операционной системой.
  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. Диспетчер задач

Диспетчер задач — [компьютерная программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) ([утилита](https://ru.wikipedia.org/wiki/%D0%A3%D1%82%D0%B8%D0%BB%D0%B8%D1%82%D0%B0)) для вывода на [экран](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BF%D0%BB%D0%B5%D0%B9) списка запущенных [процессов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) и потребляемых ими ресурсов (в частности статус, [процессорное время](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B2%D1%80%D0%B5%D0%BC%D1%8F) и потребляемая [оперативная память](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C)). В качестве дополнительных функций, диспетчер задач может предложить возможность завершить один из процессов или присвоить ему другой [приоритет](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BE%D1%80%D0%B8%D1%82%D0%B5%D1%82).

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.

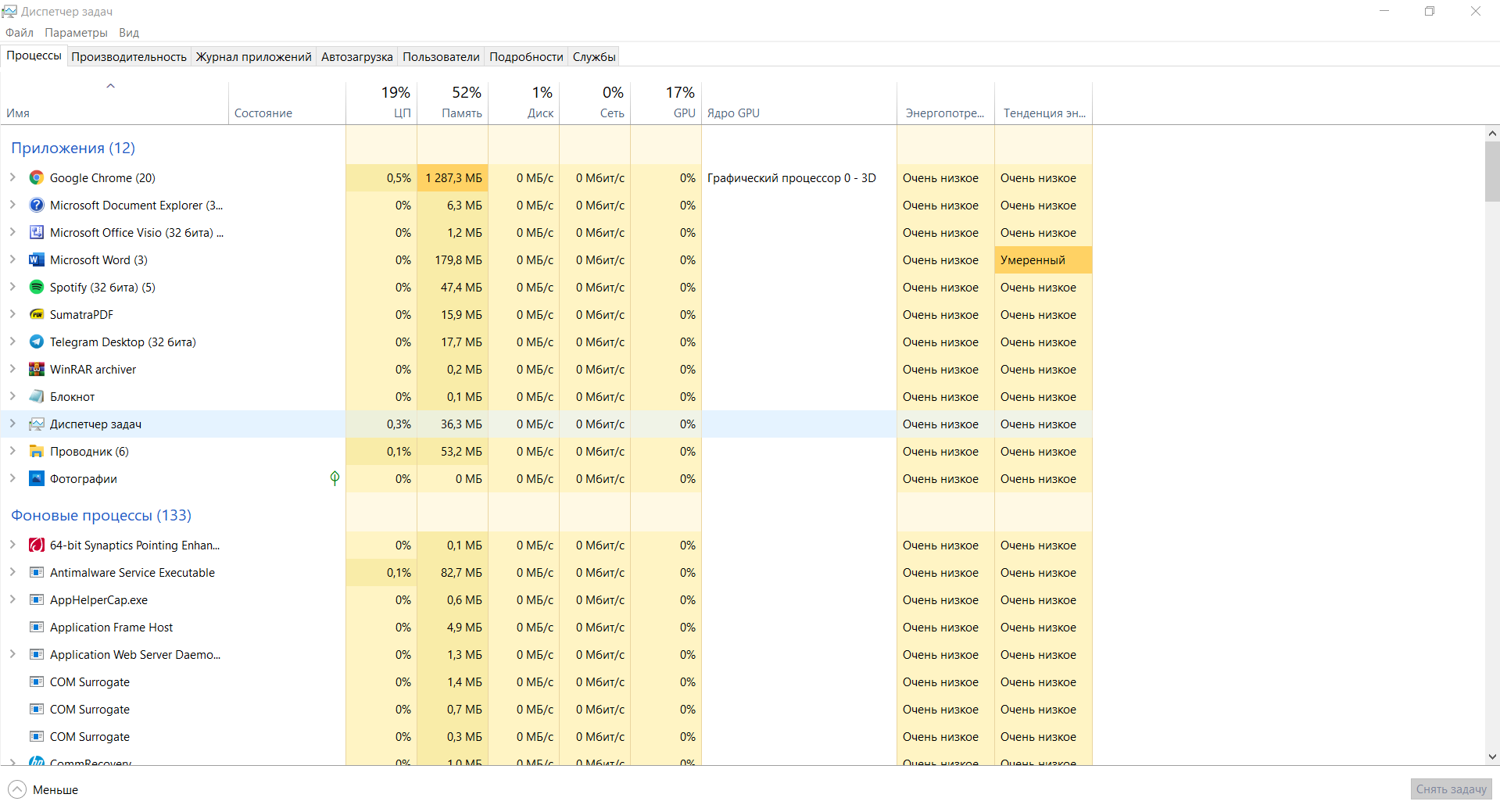


Рисунок 1.5 – интерфейс Диспетчера задач

* + 1. 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

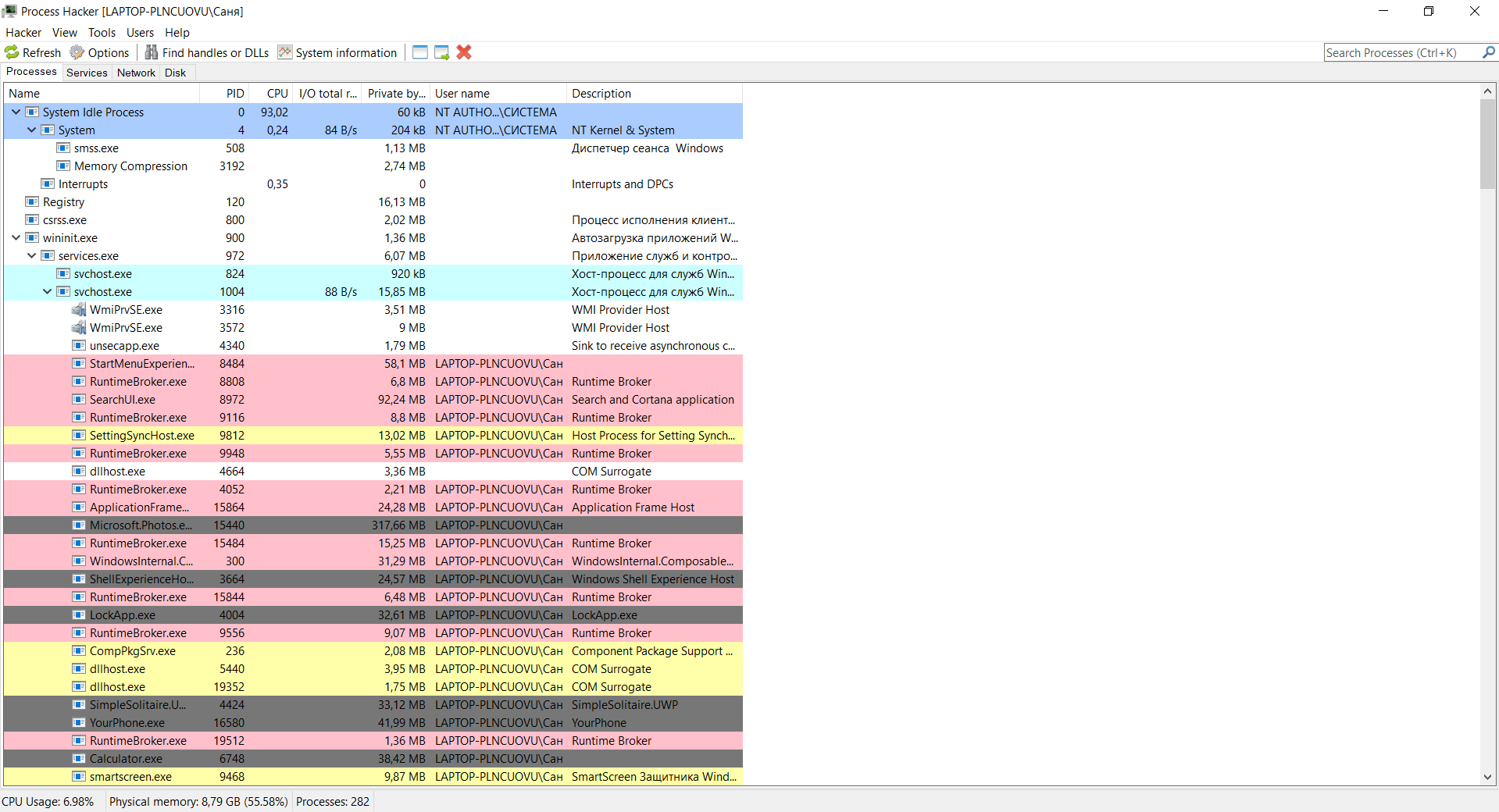


Рисунок 1.6 – интерфейс Process Hacker

* + 1. 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.

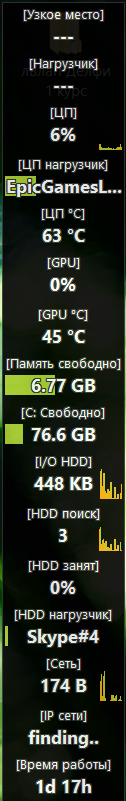


Рисунок 1.7 – интерфейс виджета Moo0 System Monitor

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

* 1. 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 2019 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 рекомендацием разработчиков данной технологии.

1. Анализ требований к программному средству и разработка функциональных требований
   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. **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.

1. Проектирование программного средства
   1. Разработка алгоритма получения значений потребляемых ресурсов ПК
      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 Б.1.

* + 1. 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 Б.2.

* 1. 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 Б.3.

* 1. 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.

1. Конструирование программного средства
   1. Проектирование интерфейса программы

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

Рассмотрим компоненты интерфейса, которые я использовал.

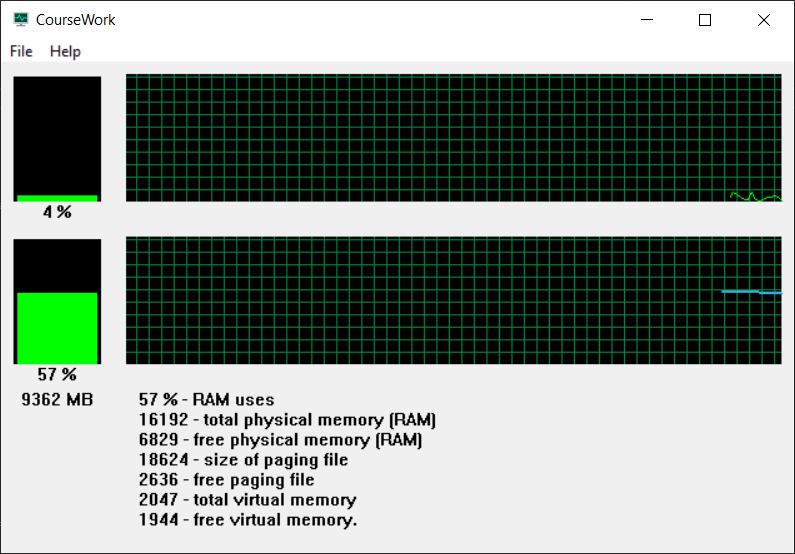


Рисунок 4.1 – интерфейс приложения

Приложение является однооконным. Стандартный размер – 650x450. В верхней части расположена панель меню с кнопками File и Help.

* 1. 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;
   1. 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 |
| ProcessList | Обработчик сообщений для окна списка процессов | LRESULT CALLBACK ProcessList  (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 |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ThreadsList | Обработчик сообщений для окна списка потоков | DWORD WINAPI ThreadProc  (LPVOID lpParam) | lpParam | Указатель, предоставляемый юзером |

* 1. Описание модуля Resourse

Модуль Resourse является модулем автогенерируемым Microsoft Visual C++ модулем. В нём хранятся константные значения, которые необходимы модулю CourseWork.

* 1. Описание модуля stdafx

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

* 1. 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.

4.9 Описание модуля с иконками для приложения

Для того, чтобы в приложении были иконки, они добавляются из кода с помощью формата ICO. Расширение .ico кодируется попиксельно с помощью массива на языке С. Таким образом, иконки можно собрать в один заголовочный файл resourses.h.

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

1. 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.

* 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 |

* 1. 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 всплывающее меню Process List | Запуск приложения, наведение курсора на меню File, запуск меню 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, наведение курсора на меню File, запуск меню 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, наведение курсора на меню File, запуск меню Process List, запуск стороннего приложения | Значения изменятся в соответствии со встроенным Диспетчером задач | Тест пройден |
| 4 | Корректность вывода ID процесса стороннего приложения | Запуск приложения, наведение курсора на меню File, запуск меню 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, наведение курсора на меню File, запуск меню Process List, запуск стороннего приложения, закрытие стороннего приложения | Значения изменятся в соответствии со встроенным Диспетчером задач | Тест пройден |

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

1. Руководство по установке и использованию

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

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

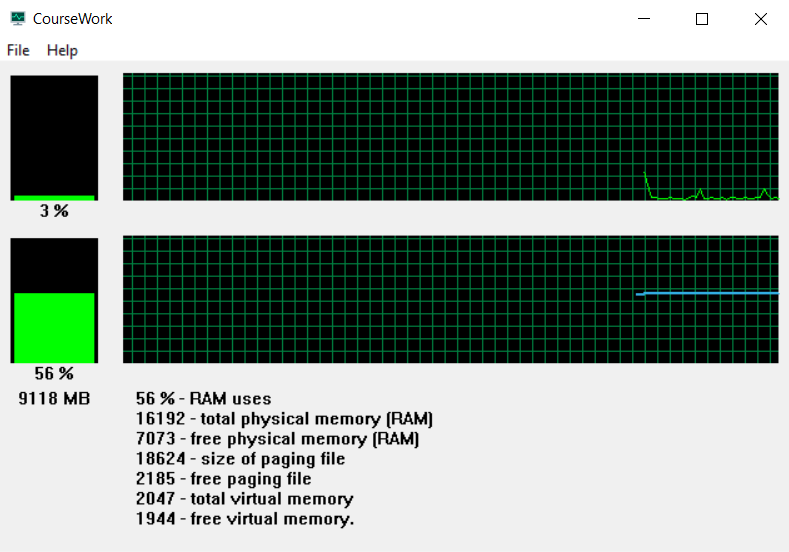


Рисунок 6.1 – Программное средство на устройстве под управлением Windows 10

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

При запуске программы отображается основное окно (рис 6.1). В верхней части окна приложения расположена панель меню (рис 6.2).



Рисунок 6.2 – панель меню

Панель инструментов можно условно разделить на следующие области:

1. Панель File;
2. Панель File;

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 6.3).

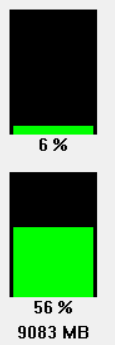


Рисунок 6.3 – диаграммы нагруженности ЦП и оперативной памяти

Правее от них так же соответственно находятся графики, которые отображают нагруженность ЦП и оперативной памяти ПК в конкретный момент времени (рис. 6.4)

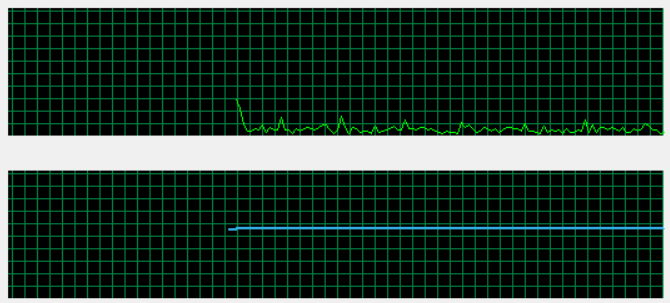


Рисунок 6.4 – графики нагруженности ЦП и оперативной памяти

В самом низу (рис. 6.5) 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

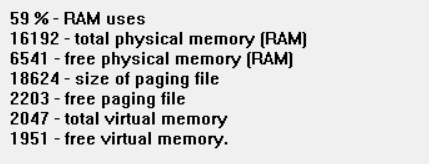


Рисунок 6.5 – значения свободной и занятой памяти разных категорий

По нажатию на кнопку Process List всплывающего меню File 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 (рисунок 6.6)

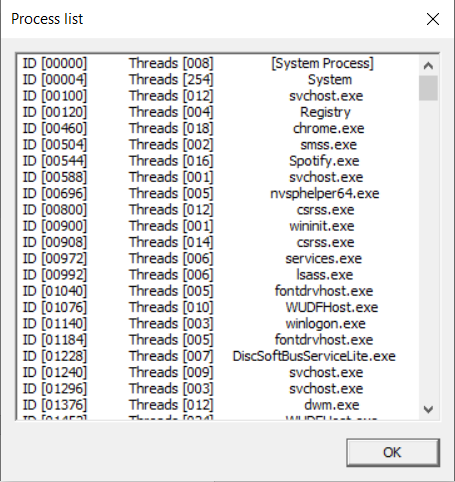


Рисунок 6.6 – окно Process List

В том же всплывающем меню имеется кнопка Exit, которая позволяет закрыть приложение.

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](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%20)

[4] ISO/IEC 14882:2011

Приложение A

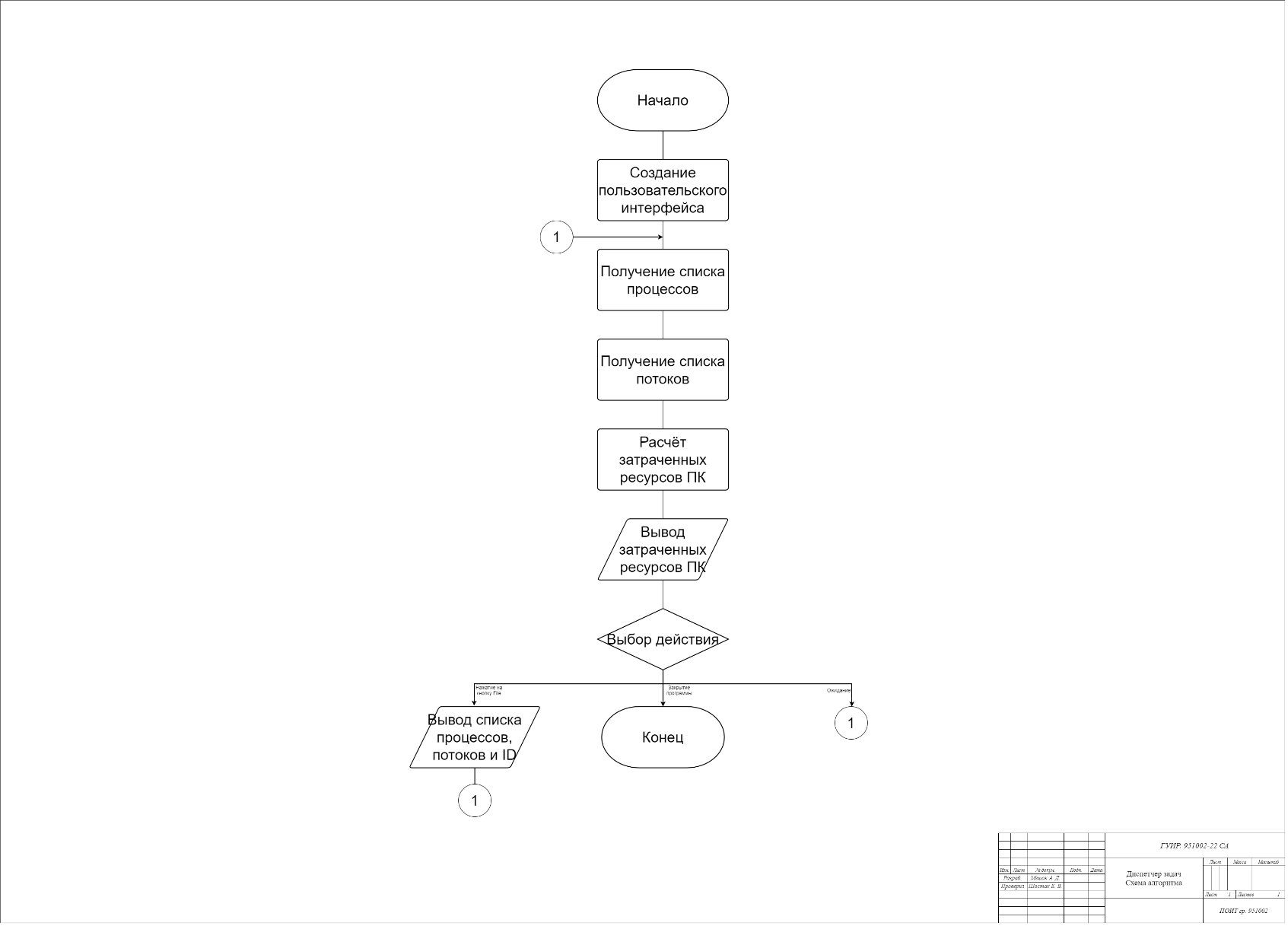


Рисунок А.1 – Схема алгоритма программы

Приложение Б

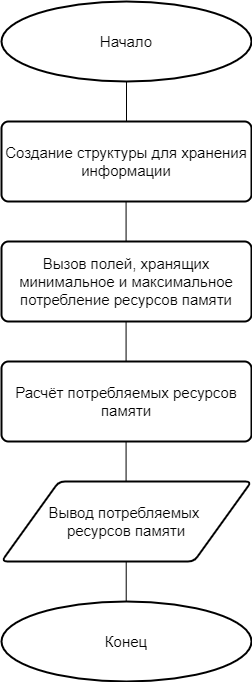


Рисунок Б.1 – Схема алгоритма вывода потребляемых ресурсов памяти

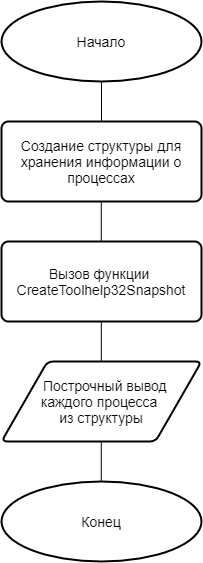


Рисунок Б.2 – Схема алгоритма вывода списка процессов

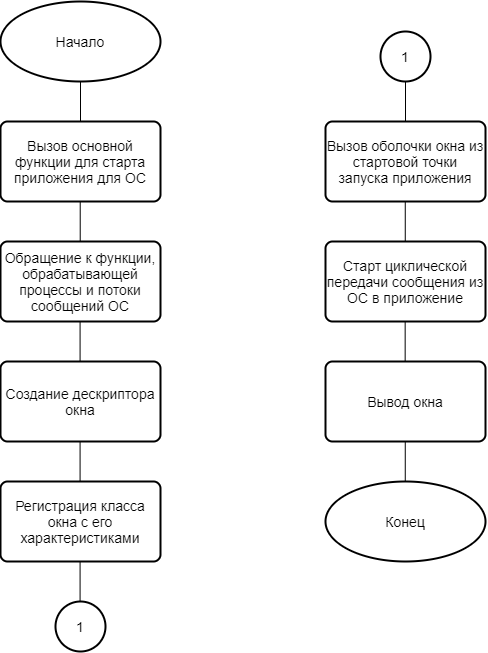


Рисунок Б.3 – Схема алгоритма создания окна в WinAPI

Приложение В

Листинг файла CourseWork.cpp

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

//

#include "stdafx.h"

#include "CourseWork.h"

#include <stdio.h>

#include <string>

#include <WinUser.h>

#include "Commdlg.h"

#include<Strsafe.h>

#include "TlHelp32.h"

#include <string.h>

#include <stdint.h>

#include <memory>

#include "TlHelp32.h" // for procces list

#define MAX\_LOADSTRING 100

#define DIV 1048576 // = 1MB

#define ID\_CPU 2000

#define ID\_MEM 2001

#define ID\_MEM2 2003

#define ID\_EDIT 2002

#define SIZE 400

// Global Variables:

HINSTANCE hInst; // current instance

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

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

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};

FILETIME idleTime;

FILETIME kernelTime;

FILETIME userTime;

FILETIME last\_idleTime;

FILETIME last\_kernelTime;

FILETIME last\_userTime;

static THREADS Thread;

// 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\_PTR CALLBACK ProcessList(HWND, UINT, WPARAM, LPARAM);

DWORD WINAPI ThreadProc(LPVOID);

int APIENTRY \_tWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPTSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place code here.

MSG msg;

HACCEL hAccelTable;

// Initialize global strings

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

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

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_COURSEWORK));

// Main message loop:

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(hInstance, MAKEINTRESOURCE(IDI\_ICON1));

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

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

//wcex.hbrBackground = CreatePatternBrush(LoadBitmap(hInstance,

MAKEINTRESOURCE(IDB\_BITMAP1)));

wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_COURSEWORK);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_ICON1));

return RegisterClassEx(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

HWND hWnd;// , edit\_cpu, edit\_mem, edit\_mem2, edit2;

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

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

300, 200, 0, 0, NULL, NULL, hInstance, NULL);

if (!hWnd)

{

return FALSE;

}

HWND edit\_cpu = CreateWindow(L"static", L"", WS\_CHILD | WS\_VISIBLE |

ES\_CENTER, 10, 110, 70, 20, hWnd,

(HMENU)ID\_CPU, hInstance, NULL);

HWND edit\_mem = CreateWindow(L"static", L"", 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, 200, hWnd, (HMENU)ID\_EDIT, hInstance, NULL);

Thread.handleThread = NULL;

Thread.ThreadFucntion = ThreadProc;

//Thread.idcTimeEdit = IDC\_EDIT1;

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);

// Parse the menu selections:

switch (wmId)

{

case IDM\_PROCESSLIST:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_PROCESSLISTBOX), hWnd,

ProcessList);

break;

case IDM\_EXIT:

DestroyWindow(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, IDC\_LIST1);

HANDLE processHandle, currentProcess;

// structure which describe procces

PROCESSENTRY32 processes;

WCHAR buffer[255];

// create processes screen

processHandle = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

processes.dwSize = sizeof(PROCESSENTRY32);

if (Process32First(processHandle, &processes))

{

do

{

wsprintfW(buffer, L" ID [%05d] Threads [%03d] %30s",

processes.th32ProcessID, processes.cntThreads,

processes.szExeFile);

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;

}

DWORD WINAPI ThreadProc(LPVOID lpParam)

{

THREADS \*ti = (THREADS\*)lpParam;

GetSystemTimes(&last\_idleTime, &last\_kernelTime, &last\_userTime);

for (;;)

{

if (GetSystemTimes(&idleTime, &kernelTime, &userTime) != 0)

{

//Sleep(100);

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, ID\_CPU, cpu\_char);

}

}

MEMORYSTATUSEX statex;

statex.dwLength = sizeof (statex);

GlobalMemoryStatusEx(&statex);

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, ID\_MEM, 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, ID\_MEM2, 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, ID\_EDIT, buffer);

InvalidateRect(Thread.handleDialog, NULL, TRUE);

Sleep(1000);

}

}

Листинг файла CourseWork.h

#pragma once

#include "resource.h"

Листинг файла Resourse.h

//{{NO\_DEPENDENCIES}}

// Microsoft Visual C++ generated include file.

// Used by CourseWork.rc

//

#define IDC\_MYICON 2

#define IDD\_COURSEWORK\_DIALOG 102

#define IDS\_APP\_TITLE 103

#define IDD\_ABOUTBOX 103

#define IDM\_ABOUT 104

#define IDM\_EXIT 105

#define IDI\_COURSEWORK 107

#define IDI\_SMALL 108

#define IDC\_COURSEWORK 109

#define IDR\_MAINFRAME 128

#define IDI\_ICON1 129

#define IDB\_BITMAP1 130

#define IDD\_PROCESSLISTBOX 131

#define IDC\_LIST1 1000

#define ID\_1 32771

#define ID\_FILE\_PROCESSLIST 32772

#define ID\_PROCESSLIST 32773

#define IDM\_PROCESSLIST 32774

#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 132

#define \_APS\_NEXT\_COMMAND\_VALUE 32775

#define \_APS\_NEXT\_CONTROL\_VALUE 1001

#define \_APS\_NEXT\_SYMED\_VALUE 110

#endif

#endif

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 01 222 ПЗ | | | | Пояснительная записка | | | | 46 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951002 222 СП | | | | Схема программы «Диспетчер задач» | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КР 1-40 01 01 222 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Диспетчер задач  Ведомость курсовой  работы |  | | | | Лист | Листов |
| Разраб. | | Мешок А.Д. |  | 22.12.20 | Т |  | |  | 46 | 46 |
| Пров. | | Шостак Е.В. |  | 22.12.20 | Кафедра ПОИТ  гр. 951002 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |