**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ **«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. В. Г. ШУХОВА»**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**ЛАБОРАТОРНАЯ РАБОТА №5**

**Дисциплина: Операционные системы**

**Тема: Операции с файлами**

Выполнил: ст. группы ВТ-31

Подкопаев Антон Валерьевич

Проверил: ст. пр. ПОВТиАС

Михелев Владимир Михайлович

**Белгород 2019**

**Цель работы:** Получение практических навыков по использованию Win32 API для работы с файлами.

**Вариант 13**

Реализовать программу, выполняющую сортировку последовательности вещественных чисел в бинарном файле. Использовать один из методов сортировки данных во внешней памяти (блочный, ленточный и т.п.). Размер файла заведомо больше объема оперативной памяти, которую может использовать приложение для сортировки.

**Файлы инициализации**

Windows дает приложениям возможность легко сохранить данные инициализации или состояния в текстовых файлах стандартного формата, называемых файлами инициализации. Самыми известными примерами файлов инициализации являются WIN.INI и SYSTEM.INI. Они используются некоторыми версиями Windows для сохранения системных настроек (например, информации о шрифтах и устройствах), а также приложениями для сохранения их собственных данных инициализации. Windows также позволяет каждому приложению создавать и поддерживать собственные файлы инициализации.

Файлы инициализации (или профильные файлы, как их иногда называют) имеют стандартный формат, описанный в документации Windows.

Каждый файл инициализации делится на секции. Каждой секции присваивается имя, заключенное в квадратные скобки:

[имя секции]

Секция содержит записи, которые состоят из двух частей, разделенных знаком =. Первая часть называется именем ключа. Вторая часть содержит строковое значение ключа. Таким образом, обобщенный формат записи выглядит так:

имя ключа = строковое значение ключа

Например, в файле WIN.TNI имеется секция для описания различных звуков, воспроизводимых системой при наступлении некоторых событий. Стандартный звуковой сигнал задается ключом SystemDefault. Строковое значение определяет имя звукового файла (например, DING.WAV) за которым следует описание:

[sounds]

SystemDefault=dmg.wav, Default Beep

В Windows NT большинство секций WIN.INI и других системных файлов инициализации отображается на реестр. Чтобы определить, какие секции файлов отображаются на реестр, загляните в раздел реестра HKEY\_LOCAL\_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\IniFileMapping. Если секция INI-файла отображается на реестр, функции работы с INI-файлами записывают данные прямо в реестр, и эти данные вообще не попадают в файлы инициализации. Механизм отображения полностью прозрачен для программиста, поэтому вы можете смело пользоваться функциями файлов инициализации и не беспокоиться о том, в какой операционной системе вы работаете, и где реально сохраняются данные — в файле или системном реестре.

Программы на C++ и Delphi могут использовать эти функции, как для ведения собственных файлов инициализации, так и для чтения или записи содержимого файла WIN.INI. Функции API для работы с файлами инициализации перечислены в следующей таблице.

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| *GetProfileInt*  *GetProfileSection*  *GetProfileString*  *GetPrivateProfileInt*  *GetPrivateProfileSection*  *GetPrivateProfileString*  *WritePrivateProfileSectiоn*  *WritePrivateProfileString*  *WriteProfileSection*  *WriteProfileString* | Читает целый параметр из файла инициализации WIN.INI  Читает секцию из файла инициализации WIN.INI  Читает строку из файла инициализации WIN.INI  Читает целый параметр из файла инициализации WIN.INI  Читает секцию из приватного файла инициализации  Читает строку из приватного файла инициализации  Записывает секцию в приватный файл инициализации  Записывает строку в приватный файл инициализации  Записывает секцию в файл инициализации WIN.INI  Записывает строку в файл инициализации WIN.INI |

**Системный реестр**

Одна из отличительных особенностей DOS и других операционных систем того же класса заключается в том, что пользователь работает с приложением, а приложения, в свою очередь, работают с данными. Иначе говоря, когда вы хотите что-нибудь сделать, вы выбираете приложение, с которым вы собираетесь работать. Затем при помощи приложения вы обращаетесь к файлам на диске, содержащим нужные данные.

Windows обеспечивает другой, документно-ориентированный подход. Вместо того чтобы выбирать приложение, вы выбираете данные, с которыми собираетесь работать. Далее операционная система выбирает нужное приложение. Например, если у вас имеется документ с изображением, которое вы хотите отредактировать, вместо запуска графического редактора можно просто открыть документ. Windows ищет в вашей системе графический редактор, способный обрабатывать графический формат документа, запускает его и открывает документ для редактирования. Благодаря новой технологии OLE документ может содержать много различных типов данных, каждый из которых может редактироваться отдельным приложением. В режиме редактирования «на месте» окно главного приложения временно занимается этими вспомогательными приложениями, что позволяет редактировать разнотипные данные документа в одном окне, предоставляющем интерфейс к нескольким приложениям. Например, если вы используете Microsoft Word для редактирования документа, содержащего электронную таблицу Excel, то при щелчке на таблице в окне Word активизируются меню и интерфейсные элементы Excel.

Windows отслеживает связи между этими объектами и документом, упрощая совместное использование объектов в нескольких документах и обеспечивая целостность ссылок даже при перемещении отдельных файлов на диске.

Но какое отношение все сказанное имеет к системному реестру?

Чтобы операционная система могла узнать, какое приложение связывается с тем или иным типом объектов, она должна располагать многочисленными данными и об объектах, и о приложениях. Системе необходимы средства уникальной регистрации и идентификации любых объектов - определяемых как системой, так и отдельными приложениями. Нужно уметь определять, какое приложение связано с конкретным объектом. Нужно знать, где на диске хранятся документы и объекты, чтобы их можно было быстро найти. Нужно следить за общими компонентами, чтобы при удалении приложения не удалялись его компоненты, используемые в других приложениях.

По мере эволюции Windows объемы и типы информации, которую приходится хранить для решения этих задач, выходят за пределы возможностей эффективного использования файлами инициализации. По этой причине Microsoft; разработала системный реестр (также называемый «регистрационной базой данных»). Реестр может содержать существенно больше данных, чем файлы инициализации, и не ограничивается строковым типом данных. Он более надежен и лучше защищен от ошибок, чем отдельные файлы инициализации, и его сложнее отредактировать или удалить. Реестр также содержит некоторые данные конфигурации системы в зависимости от операционной системы. В Windows 3.x почти все данные системной конфигурации хранились в файлах инициализации — при удалении реестра информация практически не терялась. В Windows 95 часть данных системной конфигурации хранится в реестре, но даже в случае его повреждения или удаления система может сохранить работоспособность. В Windows NT потеря реестра является катастрофой — реестр придется либо восстанавливать с диска аварийного восстановления, либо переустанавливать систему заново, поскольку Windows NT почти не хранит данные в файлах инициализации.

В документации Win32 утверждается, что файлы инициализации устарели и вам следует ограничиваться функциями реестра. Это не так. Иногда файлы инициализации оказываются более удачным решением — например, если вы хотите, чтобы пользователь мог напрямую редактировать данные конфигурации. Выбор между реестром и файлами инициализации осуществляется в соответствии с потребностями конкретного приложения.

**Архитектура реестра**

Реестр чем-то напоминает миниатюрную файловую систему. Вместо файлов он содержит параметры (values), а вместо каталогов — ключи (keys). Все вместе образует информационную иерархию. Вместо одного корневого каталога в реестре Windows 95 и Windows NT находится несколько стандартных ключей. Функциям, которым передаются дескрипторы ключей, в большинстве случаев также можно передавать константы стандартных ключей, перечисленные в следующей таблице:

|  |  |
| --- | --- |
| *HKEY\_CLASSES\_ROOT*  *HKEY\_CURRENT\_USER*  *HKEY\_LOCAL\_MACHINE*  *HKEY\_USERS*  *HKEY\_CURRENT\_CONFIG*  *HKEY\_DYN\_DATA* | Ключ предназначен для хранения информации о классах файлов и объектов. Здесь хранятся данные о связи типов документов с приложениями, а также идентификаторы классов, используемые объектами OLE. К ключу также можно обратиться через HKEY\_LOCAL\_MACHINE\SOFTWARE\Classes  В ключе хранятся данные конфигурации для текущего пользователя.  Подробная информация об аппаратной и программной конфигурации системы.  Этот ключ верхнего уровня содержит данные конфигурации программ и системы, уникальные для конкретного пользователя. Подключ HKEY\_USERS\Default содержит настройки по умолчанию для новых пользователей, а также пользователей, не вносивших собственные изменения в стандартную конфигурацию  Ключ используется для хранения общих данных конфигурации системы. |

Если запустить на выполнение редактор реестра Regedit, то в окне редактора реестра можно увидеть шесть стандартных ключей, отображенных в виде узлов верхнего уровня. Текущим элементом можно выбрать ключ HKEY\_CURRENT\_USER\Control Panel\Desktop\WindowMetrics.

Каждому элементу этой строки соответствует некоторый ключ. HKEY\_CURRENT\_USER — корневой ключ, Control Panel — подключ, Desktop — подключ ключа Control Panel, а WindowMetrics - подключ Desktop. WindowMetrics не содержит подключей, но в нем находится несколько параметров. С каждым ключом может быть связан параметр по умолчанию. У этого параметра нет имени, и если он присутствует, то всегда имеет строковый тип. Ключ WindowMetrics содержит большое количество параметров, содержащих как строковые, так и двоичные данные. Старайтесь не менять настройку системных параметров функциями реестра. Всегда используйте функции API, если они существуют (а если не существуют, скорее всего, прямое изменение параметра в реестре приведет к нежелательным результатам).

Ключи реестра могут содержать данные различных типов, приведенных в следующей таблице.

|  |  |
| --- | --- |
| **Тип данных** | **Описание** |
| *REG\_BINARY* | Двоичные данные |
| *REG\_DWORD* | 32-разрядное число |
| *REG\_DWORD\_BIGENDIAN* | 32-разрядное число, байты которого упорядочены так, что старший байт находится по младшему адресу |
| *REG\_DWORD\_LITTLE\_ENDIAN* | 32-разрядное целое, байты которого упорядочены так, что младший байт находится по младшему адресу (стандарт Intel Х86) |
| *REG\_EXPAND\_SZ* | Нерасширяемая строка окружения (см. функцию ExpandEnvironmentStrings) |
| *REG\_LINK* | Символическая ссылка |
| *REG\_MULTI\_SZ* | Список строк, разделяемых нуль-символами. Завершается двумя нуль-символами |
| *REG\_NONE* | Неопределенный тип |
| *REG\_RESOURCE\_LIST* | Список ресурсов для драйвера устройства |
| *REG\_SZ* | Строка, завершенная нуль-символом |

Win32 содержит большое количество функций для работы с реестром. Эти функции перечислены в следующей таблице.

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| *RegCIoseKey* | Закрывает открытый ключ |
| *RegConnectRegistry* | Открывает ключ в удаленной системе |
| *RegCreateKey,RegCreateKeyEx* | Создает новый ключ |
| *RegDeleteKey* | Удаляет ключ |
| *RegDeleteValue*  *RegEnumKey, RegEnumKeyEx* | Удаляет параметр  Перебирает подключи заданного ключа |
| *RegEnumValue* | Перебирает параметры ключа |
| *RegFlushKey* | Обеспечивает запись на диск изменений в ключе |
| *RegGetKeySecunty* | Получает данные безопасности для ключа |
| *RegLoadKey* | Загружает информацию реестра, сохраненную в файле |
| *RegNotifyChangeKeyValue* | Оповещает приложение об изменениях значения или других атрибутов ключа |
| *RegOpenKey,RegOpenKeyEx* | Открывает ключ |
| *RegQueryInfoKey* | Получает информацию о ключе |
| *RegQueryValue,RegQueryValueEx* | Читает значение параметра |
| *RegReplaceKey* | Замещает данные реестра данными, сохраненными в файле |
| *RegRestoreKey* | Восстанавливает данные реестра по данным, сохраненным в файле |
| *RegSaveKey* | Сохраняет данные реестра в файле |
| *RegSetKeySecurity* | Задает атрибуты безопасности для ключа |
| *RegSetValue, RegSetValueEx* | Задает значение параметра |
| *RegUnloadKey* | Выгружает заданный ключ и все его подключи из реестра |

MainWindow.xaml.cs

using Microsoft.Win32;

using Microsoft.Win32.SafeHandles;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Forms;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Threading;

using static WpfApp20.winapi;

namespace WpfApp20

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

///

public partial class MainWindow : Window

{

class FileBlock

{

public string filename;

public int size = 0;

public IntPtr fileHandle = IntPtr.Zero;

public int MAXValue, MINValue;

public List<FileBlock> sub\_blocks;

public FileBlock(string filename, IntPtr fileHandle, int minvalue, int maxvalue)

{

this.filename = filename; this.size = 0; this.fileHandle = fileHandle; this.MAXValue = maxvalue; this.MINValue = minvalue;

}

}

//string example0 = ReadRegKey(HKEY\_LOCAL\_MACHINE, @"SOFTWARE\Xiph.Org\Open Codecs", "MediaDescNum");

int partition<T>(T[] m, int a, int b) where T : IComparable<T>

{

int i = a;

for (int j = a; j <= b; j++) // просматриваем с a по b

{

if (m[j].CompareTo(m[b]) <= 0) // если элемент m[j] не превосходит m[b],

{

T t = m[i]; // меняем местами m[j] и m[a], m[a+1], m[a+2] и так далее...

m[i] = m[j]; // то есть переносим элементы меньшие m[b] в начало,

m[j] = t; // а затем и сам m[b] «сверху»

i++; // таким образом последний обмен: m[b] и m[i], после чего i++

}

}

return i - 1; // в индексе i хранится <новая позиция элемента m[b]> + 1 }

void quicksort<T>(T[] m, int a, int b) where T : IComparable<T>// a - начало подмножества, b - конец

{ // для первого вызова: a = 0, b = <элементов в массиве> - 1

if (a >= b) return;

int c = partition(m, a, b);

quicksort(m, a, c - 1);

quicksort(m, c + 1, b);

}

private static string ReadRegKey(UIntPtr rootKey, string keyPath, string valueName)

{

UIntPtr hKey = UIntPtr.Zero;

if (RegOpenKeyEx(rootKey, keyPath, 0, (int)RegistrySecurity.KEY\_READ, out hKey) == 0)

{

int size = 1024;

RegistryValueKind type = RegistryValueKind.String;

string keyValue = null;

StringBuilder keyBuffer = new StringBuilder((int)size);

//ObjectHandleExtensions.ToIntPtr(keyBuffer);

if (RegQueryValueEx(hKey, valueName, 0, ref type, keyBuffer, ref size) == 0)

keyValue = keyBuffer.ToString();

//RegCloseKey(hKey);

return (keyValue);

}

uint lasterror = GetLastError();

return (null); // Return null if the value could not be read

}

static IntPtr infileHANDLE, outfileHANDLE;

string infileName = ReadRegKey(HKEY\_LOCAL\_MACHINE, @"SOFTWARE\lab5", "IN\_File"), outfileName = ReadRegKey(HKEY\_LOCAL\_MACHINE, @"SOFTWARE\lab5", "OUT\_File");

int filecounter = 0;

int outfilePointer = 0;

void RecursiveBlockSort(string InFileName, string OutFileName, int sizeToRead, int parts, ulong RAMAviable, bool firstcall=true)

{

infileHANDLE = CreateFile(

(string)InFileName,

winapi.FileAccess.FILE\_GENERIC\_READ,

winapi.FileShare.Write | winapi.FileShare.Read | winapi.FileShare.Delete,

IntPtr.Zero,

System.IO.FileMode.Open,

winapi.FileAttributes.Overlapped,

IntPtr.Zero

);

byte[] buf = new byte[RAMAviable];

List<FileBlock> fileblocks = new List<FileBlock>();

long fileSize;

GetFileSizeEx(infileHANDLE, out fileSize);

Dispatcher.Invoke(() => logbox.Text += string.Format("FreeBytesAvailable = {0} < fileSize \* 2 = {1}\r\n", FreeBytesAvailable, (ulong)fileSize \* 2));

if (FreeBytesAvailable > (ulong)fileSize \* 2)

{

uint readedbyte = 0;

IntPtr myEventHandle = CreateEvent(IntPtr.Zero, false, true, "MyEvent");

OVERLAPPED overlapped = new OVERLAPPED();

overlapped.EventHandle = myEventHandle;

overlapped.Offset = 0;

ReadFile(infileHANDLE, buf, 1, ref readedbyte, ref overlapped);

int MAXvalue = buf[0], MINvalue = buf[0];

overlapped.Offset = 0;

for (overlapped.Offset = 0; overlapped.Offset < fileSize; overlapped.Offset += 1)

{

ReadFile(infileHANDLE, buf, 1, ref readedbyte, ref overlapped);

if (buf[0] > MAXvalue) MAXvalue = buf[0];

if (buf[0] < MINvalue) MINvalue = buf[0];

}

float k = (float)(MAXvalue - MINvalue) / parts;

MINvalue--;

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

{

fileblocks.Add(new FileBlock(String.Format(@"tmp{0}.txt", filecounter), CreateFile(

String.Format(@"tmp{0}.txt", filecounter++),

winapi.FileAccess.GenericAll,

winapi.FileShare.Write | winapi.FileShare.Read | winapi.FileShare.Delete,

IntPtr.Zero,

System.IO.FileMode.Create,

winapi.FileAttributes.Overlapped,

IntPtr.Zero

), (int)Math.Ceiling(MINvalue + k \* i + 1), (int)Math.Ceiling(MINvalue + k \* (i + 1))));

}

fileblocks[fileblocks.Count - 1].MAXValue++;

for (int i = 0; i < fileSize; i += 1)

{

overlapped.Offset = (uint)i;

ReadFile(infileHANDLE, buf, (uint)1, ref readedbyte, ref overlapped);

for (int j = 0; j < fileblocks.Count; j++)

{

if ((fileblocks[j].MINValue <= buf[0]) && (fileblocks[j].MAXValue >= buf[0]))

{

overlapped.Offset = (uint)fileblocks[j].size++;

WriteFile(fileblocks[j].fileHandle, buf, 1, out readedbyte, ref overlapped);

break;

}

}

}

CloseHandle(infileHANDLE);

for (int i = 0; i < fileblocks.Count; i++)

{

if (fileblocks[i].size <= (int)RAMAviable)

{

overlapped.Offset = 0;

ReadFile(fileblocks[i].fileHandle, buf, (uint)fileblocks[i].size, ref readedbyte, ref overlapped);

quicksort<byte>(buf, 0, fileblocks[i].size - 1);

overlapped.Offset = (uint)outfilePointer;

WriteFile(outfileHANDLE, buf, (uint)fileblocks[i].size, out readedbyte, ref overlapped);

outfilePointer += fileblocks[i].size;

CloseHandle(fileblocks[i].fileHandle);

DeleteFile(String.Format(fileblocks[i].filename, i));

}

else

{

int duplicates = 0;

byte[] firstelement = new byte[1];

overlapped.Offset = 0;

ReadFile(fileblocks[i].fileHandle, firstelement, 1, ref readedbyte, ref overlapped);

for (int j = 0; j < fileblocks[i].size; j++)

{

ReadFile(fileblocks[i].fileHandle, buf, 1, ref readedbyte, ref overlapped);

if (buf[0] == firstelement[0])

{

duplicates++;

}

overlapped.Offset++;

}

if (fileblocks[i].size == duplicates)

{

overlapped.Offset = 0;

for (int j = 0; j < fileblocks[i].size; j++)

{

overlapped.Offset = (uint)outfilePointer;

WriteFile(outfileHANDLE, firstelement, 1, out readedbyte, ref overlapped);

outfilePointer++;

}

CloseHandle(fileblocks[i].fileHandle);

DeleteFile(String.Format(fileblocks[i].filename, i));

//break;

}

else

{

DeleteFile(String.Format(InFileName, i));

RecursiveBlockSort(fileblocks[i].filename, OutFileName, sizeToRead, parts, RAMAviable, false);

CloseHandle(fileblocks[i].fileHandle);

DeleteFile(String.Format(fileblocks[i].filename, i));

}

}

}

}

else

{

DialogResult result = System.Windows.Forms.MessageBox.Show(

"Для сортировки не достаточно свободной памяти на жёстком диске, освободите память и попробуйте запустить сортировку заново.",

"Ошибка!",

MessageBoxButtons.OK,

MessageBoxIcon.Error,

MessageBoxDefaultButton.Button1

);

Environment.Exit(0);

}

}

async void task()

{

filecounter = 0;

MEMORYSTATUSEX mEMORYSTATUSEX = new MEMORYSTATUSEX();

GlobalMemoryStatusEx(mEMORYSTATUSEX);

ulong RAMSize = 0;

try

{

RAMSize = ulong.Parse(ReadRegKey(HKEY\_LOCAL\_MACHINE, @"SOFTWARE\lab5", "RAM\_SIZE"));

Dispatcher.Invoke(() => logbox.Text += string.Format("Искуственное ограничение оперативный памяти до {0} байт!\r\n", RAMSize));

}

catch (Exception) { }

Dispatcher.Invoke(() => logbox.Text += string.Format("\* Запуск алгоритма блочной сортировки \*\r\n"));

Task.Run(() =>

{

RecursiveBlockSort((string)infileName, (string)outfileName, 10, 2, (ulong)(RAMSize == 0 ? mEMORYSTATUSEX.ullAvailPhys : RAMSize));

long fileSize;

bool result = GetFileSizeEx(outfileHANDLE, out fileSize);

OVERLAPPED overlapped = new OVERLAPPED();

byte[] buf = new byte[fileSize];

IntPtr myEventHandle = CreateEvent(IntPtr.Zero, false, true, "MyEvent");

overlapped.EventHandle = myEventHandle;

overlapped.Offset = 0;

uint readedbytes = 0;

ReadFile(outfileHANDLE, buf, (uint)fileSize, ref readedbytes, ref overlapped);

Dispatcher.Invoke(() =>

{

for (int i = 0; i < fileSize; i++) listBox2.Items.Add(new ListBoxItem() { Content = buf[i].ToString() });

if (fileSize >= 254) listBox2.Items.Add(new ListBoxItem() { Content = "..." });

});

CloseHandle(myEventHandle);

});

}

private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

DialogResult result = System.Windows.Forms.MessageBox.Show(

"Очистить записи программы в реестре?",

"Сообщение",

MessageBoxButtons.YesNo,

MessageBoxIcon.Information,

MessageBoxDefaultButton.Button1

);

if (result == System.Windows.Forms.DialogResult.Yes)

{

RegDeleteKeyA(HKEY\_LOCAL\_MACHINE, @"SOFTWARE\lab5");

}

}

ulong FreeBytesAvailable;

ulong TotalNumberOfBytes;

ulong TotalNumberOfFreeBytes;

public MainWindow()

{

InitializeComponent();

infileHANDLE = CreateFile(

(string)infileName,

winapi.FileAccess.FILE\_GENERIC\_READ,

winapi.FileShare.Write | winapi.FileShare.Read | winapi.FileShare.Delete,

IntPtr.Zero,

System.IO.FileMode.Open,

winapi.FileAttributes.Overlapped,

IntPtr.Zero

);

outfileHANDLE = CreateFile(

(string)outfileName,

winapi.FileAccess.GenericRead | winapi.FileAccess.GenericWrite,

winapi.FileShare.Write | winapi.FileShare.Read | winapi.FileShare.Delete,

IntPtr.Zero,

System.IO.FileMode.Create,

winapi.FileAttributes.Overlapped,

IntPtr.Zero

);

long fileSize;

bool result = GetFileSizeEx(infileHANDLE, out fileSize);

OVERLAPPED overlapped = new OVERLAPPED();

byte[] buf = new byte[fileSize];

IntPtr myEventHandle = CreateEvent(IntPtr.Zero, false, true, "MyEvent");

overlapped.EventHandle = myEventHandle;

overlapped.Offset = 0;

uint readedbytes = 0;

ReadFile(infileHANDLE, buf, (uint)fileSize, ref readedbytes, ref overlapped);

for (int i = 0; i < fileSize; i++) listBox1.Items.Add(new ListBoxItem() { Content = buf[i].ToString() });

if (fileSize >= 254) listBox1.Items.Add(new ListBoxItem() { Content = "..." });

CloseHandle(myEventHandle);

logbox.Text += "Входное имя файла = " + infileName+"\n";

logbox.Text += "Выходное имя файла = " + outfileName + "\n";

Task.Run(() =>

{

while (true)

{

string tom = System.Reflection.Assembly.GetEntryAssembly().Location.Split("\\".ToCharArray())[0];

Dispatcher.Invoke(() => logbox.Text += string.Format("Рабочий диск {0} \r\n", tom));

bool success = GetDiskFreeSpaceEx(tom + "\\", out FreeBytesAvailable, out TotalNumberOfBytes, out TotalNumberOfFreeBytes);

if (!success) throw new System.ComponentModel.Win32Exception();

Console.WriteLine((FreeBytesAvailable / TotalNumberOfBytes \* 100));

Dispatcher.Invoke(() => progressbar.Value = (100 - (float)FreeBytesAvailable / (float)TotalNumberOfBytes \* 100));

Dispatcher.Invoke(() => label2.Content = (100 - (float)FreeBytesAvailable / (float)TotalNumberOfBytes \* 100).ToString()+"%");

Dispatcher.Invoke(() => logbox.Text += string.Format("{0} кластеров оперативной памяти доступно!\r\n", FreeBytesAvailable));

if (FreeBytesAvailable < 5)

{

Dispatcher.Invoke(() => logbox.Text += "Недостаточно памяти!\r\n");

//Console.WriteLine("Недостаточно памяти!");

}

Thread.Sleep(1000);

}

});

}

private void button\_Click(object sender, RoutedEventArgs e)

{

task();

}

}

}

winapi.cs

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace WpfApp20

{

public static class winapi

{

public struct OVERLAPPED

{

public UIntPtr Internal;

public UIntPtr InternalHigh;

public uint Offset;

public uint OffsetHigh;

public IntPtr EventHandle;

}

[Flags]

public enum FileAccess : uint

{

//

// Standart Section

//

AccessSystemSecurity = 0x1000000, // AccessSystemAcl access type

MaximumAllowed = 0x2000000, // MaximumAllowed access type

Delete = 0x10000,

ReadControl = 0x20000,

WriteDAC = 0x40000,

WriteOwner = 0x80000,

Synchronize = 0x100000,

StandardRightsRequired = 0xF0000,

StandardRightsRead = ReadControl,

StandardRightsWrite = ReadControl,

StandardRightsExecute = ReadControl,

StandardRightsAll = 0x1F0000,

SpecificRightsAll = 0xFFFF,

FILE\_READ\_DATA = 0x0001, // file & pipe

FILE\_LIST\_DIRECTORY = 0x0001, // directory

FILE\_WRITE\_DATA = 0x0002, // file & pipe

FILE\_ADD\_FILE = 0x0002, // directory

FILE\_APPEND\_DATA = 0x0004, // file

FILE\_ADD\_SUBDIRECTORY = 0x0004, // directory

FILE\_CREATE\_PIPE\_INSTANCE = 0x0004, // named pipe

FILE\_READ\_EA = 0x0008, // file & directory

FILE\_WRITE\_EA = 0x0010, // file & directory

FILE\_EXECUTE = 0x0020, // file

FILE\_TRAVERSE = 0x0020, // directory

FILE\_DELETE\_CHILD = 0x0040, // directory

FILE\_READ\_ATTRIBUTES = 0x0080, // all

FILE\_WRITE\_ATTRIBUTES = 0x0100, // all

//

// Generic Section

//

GenericRead = 0x80000000,

GenericWrite = 0x40000000,

GenericExecute = 0x20000000,

GenericAll = 0x10000000,

SPECIFIC\_RIGHTS\_ALL = 0x00FFFF,

FILE\_ALL\_ACCESS =

StandardRightsRequired |

Synchronize |

0x1FF,

FILE\_GENERIC\_READ =

StandardRightsRead |

FILE\_READ\_DATA |

FILE\_READ\_ATTRIBUTES |

FILE\_READ\_EA |

Synchronize,

FILE\_GENERIC\_WRITE =

StandardRightsWrite |

FILE\_WRITE\_DATA |

FILE\_WRITE\_ATTRIBUTES |

FILE\_WRITE\_EA |

FILE\_APPEND\_DATA |

Synchronize,

FILE\_GENERIC\_EXECUTE =

StandardRightsExecute |

FILE\_READ\_ATTRIBUTES |

FILE\_EXECUTE |

Synchronize

}

[Flags]

public enum FileShare : uint

{

/// <summary>

///

/// </summary>

None = 0x00000000,

/// <summary>

/// Enables subsequent open operations on an object to request read access.

/// Otherwise, other processes cannot open the object if they request read access.

/// If this flag is not specified, but the object has been opened for read access, the function fails.

/// </summary>

Read = 0x00000001,

/// <summary>

/// Enables subsequent open operations on an object to request write access.

/// Otherwise, other processes cannot open the object if they request write access.

/// If this flag is not specified, but the object has been opened for write access, the function fails.

/// </summary>

Write = 0x00000002,

/// <summary>

/// Enables subsequent open operations on an object to request delete access.

/// Otherwise, other processes cannot open the object if they request delete access.

/// If this flag is not specified, but the object has been opened for delete access, the function fails.

/// </summary>

Delete = 0x00000004

}

public enum CreationDisposition : uint

{

/// <summary>

/// Creates a new file. The function fails if a specified file exists.

/// </summary>

New = 1,

/// <summary>

/// Creates a new file, always.

/// If a file exists, the function overwrites the file, clears the existing attributes, combines the specified file attributes,

/// and flags with FILE\_ATTRIBUTE\_ARCHIVE, but does not set the security descriptor that the SECURITY\_ATTRIBUTES structure specifies.

/// </summary>

CreateAlways = 2,

/// <summary>

/// Opens a file. The function fails if the file does not exist.

/// </summary>

OpenExisting = 3,

/// <summary>

/// Opens a file, always.

/// If a file does not exist, the function creates a file as if dwCreationDisposition is CREATE\_NEW.

/// </summary>

OpenAlways = 4,

/// <summary>

/// Opens a file and truncates it so that its size is 0 (zero) bytes. The function fails if the file does not exist.

/// The calling process must open the file with the GENERIC\_WRITE access right.

/// </summary>

TruncateExisting = 5

}

[Flags]

public enum FileAttributes : uint

{

Readonly = 0x00000001,

Hidden = 0x00000002,

System = 0x00000004,

Directory = 0x00000010,

Archive = 0x00000020,

Device = 0x00000040,

Normal = 0x00000080,

Temporary = 0x00000100,

SparseFile = 0x00000200,

ReparsePoint = 0x00000400,

Compressed = 0x00000800,

Offline = 0x00001000,

NotContentIndexed = 0x00002000,

Encrypted = 0x00004000,

Write\_Through = 0x80000000,

Overlapped = 0x40000000,

NoBuffering = 0x20000000,

RandomAccess = 0x10000000,

SequentialScan = 0x08000000,

DeleteOnClose = 0x04000000,

BackupSemantics = 0x02000000,

PosixSemantics = 0x01000000,

OpenReparsePoint = 0x00200000,

OpenNoRecall = 0x00100000,

FirstPipeInstance = 0x00080000

}

[StructLayout(LayoutKind.Explicit, Size = 8)]

public struct FILE\_SEGMENT\_ELEMENT

{

[FieldOffset(0)]

public IntPtr Buffer;

[FieldOffset(0)]

public UInt64 Alignment;

}

[DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]

public static extern IntPtr CreateFile(

[MarshalAs(UnmanagedType.LPTStr)] string filename,

[MarshalAs(UnmanagedType.U4)] FileAccess fileAccess,

[MarshalAs(UnmanagedType.U4)] FileShare fileShare,

IntPtr securityAttributes, // optional SECURITY\_ATTRIBUTES struct or IntPtr.Zero

[MarshalAs(UnmanagedType.U4)] FileMode creationDisposition,

[MarshalAs(UnmanagedType.U4)] FileAttributes flagsAndAttributes,

IntPtr templateFile);

[DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]

public static extern IntPtr CreateFile(

[MarshalAs(UnmanagedType.LPTStr)] string filename,

[MarshalAs(UnmanagedType.U4)] FileAccess fileAccess,

[MarshalAs(UnmanagedType.U4)] FileShare fileShare,

IntPtr securityAttributes, // optional SECURITY\_ATTRIBUTES struct or IntPtr.Zero

[MarshalAs(UnmanagedType.U4)] uint creationDisposition,

[MarshalAs(UnmanagedType.U4)] FileAttributes flagsAndAttributes,

IntPtr templateFile);

[DllImport("kernel32.dll")]

public static extern IntPtr CreateEvent(IntPtr lpEventAttributes, bool bManualReset, bool bInitialState, string lpName);

[DllImport("kernel32.dll", SetLastError = true)]

[return: MarshalAs(UnmanagedType.Bool)]

public static extern bool CloseHandle(IntPtr hObject);

[DllImport("kernel32.dll")]

public static extern bool ReadFile(IntPtr hFile, byte[]

aSegementArray, uint nNumberOfBytesToRead, [In] ref uint lpReserved,

[In] ref OVERLAPPED lpOverlapped);

[DllImport("kernel32.dll")]

public static extern bool WriteFile(IntPtr hFile, byte[] lpBuffer,

uint nNumberOfBytesToWrite, out uint lpNumberOfBytesWritten,

[In] ref OVERLAPPED lpOverlapped);

[DllImport("kernel32.dll")]

public static extern bool GetFileSizeEx(IntPtr hFile, out long lpFileSize);

[DllImport("kernel32.dll", SetLastError = true, CharSet = CharSet.Auto)]

[return: MarshalAs(UnmanagedType.Bool)]

public static extern bool GetDiskFreeSpaceEx(string lpDirectoryName,

out ulong lpFreeBytesAvailable,

out ulong lpTotalNumberOfBytes,

out ulong lpTotalNumberOfFreeBytes);

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]

public class MEMORYSTATUSEX

{

public uint dwLength;

public uint dwMemoryLoad;

public ulong ullTotalPhys;

public ulong ullAvailPhys;

public ulong ullTotalPageFile;

public ulong ullAvailPageFile;

public ulong ullTotalVirtual;

public ulong ullAvailVirtual;

public ulong ullAvailExtendedVirtual;

public MEMORYSTATUSEX()

{

this.dwLength = (uint)Marshal.SizeOf(typeof(MEMORYSTATUSEX));

}

}

[DllImport("kernel32.dll", SetLastError = true)]

[return: MarshalAs(UnmanagedType.Bool)]

public static extern bool DeleteFile(string lpFileName);

[return: MarshalAs(UnmanagedType.Bool)]

[DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]

public static extern bool GlobalMemoryStatusEx([In, Out] MEMORYSTATUSEX lpBuffer);

[DllImport("advapi32.dll", CharSet = CharSet.Unicode, EntryPoint = "RegDeleteKeyA", SetLastError = true)]

public static extern int RegDeleteKeyA(UIntPtr hKey, [MarshalAs(UnmanagedType.LPStr)] string subKey);

public enum RegistrySecurity

{

KEY\_ALL\_ACCESS = 0xF003F,

KEY\_CREATE\_LINK = 0x0020,

KEY\_CREATE\_SUB\_KEY = 0x0004,

KEY\_ENUMERATE\_SUB\_KEYS = 0x0008,

KEY\_EXECUTE = 0x20019,

KEY\_NOTIFY = 0x0010,

KEY\_QUERY\_VALUE = 0x0001,

KEY\_READ = 0x20019,

KEY\_SET\_VALUE = 0x0002,

KEY\_WOW64\_32KEY = 0x0200,

KEY\_WOW64\_64KEY = 0x0100,

KEY\_WRITE = 0x20006,

}

[DllImport("advapi32.dll", SetLastError = true)]

public static extern uint RegQueryValueEx(

UIntPtr hKey,

string lpValueName,

int lpReserved,

ref RegistryValueKind lpType,

[In,Out] StringBuilder lpData,

ref int lpcbData);

[DllImport("advapi32.dll", CharSet = CharSet.Auto, SetLastError = true)]

public static extern int RegOpenKeyEx(

UIntPtr hKey,

string subKey,

int ulOptions,

int samDesired,

out UIntPtr hkResult);

public static UIntPtr HKEY\_LOCAL\_MACHINE = new UIntPtr(0x80000002u);

public static UIntPtr HKEY\_CURRENT\_USER = new UIntPtr(0x80000001u);

[DllImport("kernel32.dll")]

public static extern uint GetLastError();

}

}

Изображение выглядит как снимок экрана, монитор, внутренний, компьютер

Автоматически созданное описание