МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»

(БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Дисциплина: Операционные системы

Лабораторная работа № 5

**Операции с файлами**

|  |  |
| --- | --- |
|  | Выполнила: ст. группы ПВ-31  Зановская А.И.  Проверил: Михелев В.М. |

Белгород

2018

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

**Задание**

* выполнить с помощью асинхронных файловых операций следующие задания:

|  |  |
| --- | --- |
| 12 | Реализовать программу, считывающую информацию об установленных в системе на сменных дисках (НГМД, CD-ROM и т.п.). Если в приводе присутствует носитель, то определить тип и характеристики файловой системы. Отчет по работе программы вывести в файл. |
|  |  |

* настройки приложения должны храниться в реестре, в ветке HKCU\Software,
* приложение должно определять свободное место на диске и проверять корректность всех файловых операций,

**Текст программы**

#include "stdafx.h"

#include <Windows.h>

#include <conio.h>

#include <iostream>

#include <fstream>

#include <locale>

#include <string.h>

#include <string>

#include <tchar.h>

#ifdef \_UNICODE

# define \_tcout wcout

#else

# define \_tcout cout

#endif // \_UNICODE

using namespace std;

void volumeInformation(int i, HANDLE h, DWORD &size, OVERLAPPED ovl);

void driveType(string s, HANDLE h, DWORD &size, OVERLAPPED ovl);

DWORD logicalDrivers(char\* fileName);

LPWSTR ConvertToLPWSTR(const std::string& s);

int createKey(char \*s);

string toHex(DWORD a);

int main()

{

setlocale(LC\_ALL, "rus");

char path[MAX\_PATH];

int t;

if (t = createKey(path) != 0)

return 0;

logicalDrivers(path);

system("pause");

}

void volumeInformation(int i, HANDLE h, DWORD &size, OVERLAPPED ovl)

{

char VolumeNameBuffer[100];

char FileSystemNameBuffer[100];

DWORD FileSystemFlags;

unsigned long VolumeSerialNumber;

char dd[4];

dd[0] = char(65 + i); dd[1] = ':'; dd[2] = '\\'; dd[3] = 0;

BOOL GetVolumeInformationFlag = GetVolumeInformationA(

dd,

VolumeNameBuffer,

100,

&VolumeSerialNumber,

NULL, //&MaximumComponentLength,

&FileSystemFlags,

FileSystemNameBuffer,

100

);

if (GetVolumeInformationFlag != 0)

{

string t = " Volume Name " + (string)VolumeNameBuffer + '\n';

WriteFile(h, t.c\_str(), (t).size(), &size, &ovl);

size += (t).size();

t = " Volume Serial Number " + to\_string(VolumeSerialNumber) + '\n';

WriteFile(h, t.c\_str(), (t).size(), &size, &ovl);

size += (t).size();

t = " File System " + (string)FileSystemNameBuffer + '\n';

WriteFile(h, t.c\_str(), (t).size(), &size, &ovl);

size += (t).size();

t = " System flags ";

t += "0x" + toHex(FileSystemFlags) + '\n';

WriteFile(h, t.c\_str(), (t).size(), &size, &ovl);

size += (t).size();

string tmp = "";

if (t[9] == '2')

tmp += "FILE\_CASE\_PRESERVED\_NAMES\n";

if (t[9] == '1')

tmp += "FILE\_CASE\_SENSITIVE\_SEARCH\n";

if (t[8] == '1')

tmp += "FILE\_FILE\_COMPRESSION\n";

if (t[5] == '4')

tmp += "FILE\_NAMED\_STREAMS\n";

if (t[9] == '8')

tmp += "FILE\_PERSISTENT\_ACLS\n";

if (t[5] == '8')

tmp += "FILE\_READ\_ONLY\_VOLUME\n";

if (t[4] == '1')

tmp += "FILE\_SEQUENTIAL\_WRITE\_ONCE\n";

if (t[5] == '2')

tmp += "FILE\_SUPPORTS\_ENCRYPTION\n";

if (t[4] == '8')

tmp += "FILE\_SUPPORTS\_EXTENDED\_ATTRIBUTES\n";

if (t[4] == '4')

tmp += "FILE\_SUPPORTS\_HARD\_LINKS\n";

if (t[5] == '1')

tmp += "FILE\_SUPPORTS\_OBJECT\_IDS\n";

if (t[3] == '1')

tmp += "FILE\_SUPPORTS\_OPEN\_BY\_FILE\_ID\n";

if (t[8] == '8')

tmp += "FILE\_SUPPORTS\_REPARSE\_POINTS\n";

if (t[8] == '4')

tmp += "FILE\_SUPPORTS\_SPARSE\_FILES\n";

if (t[4] == '2')

tmp += "FILE\_SUPPORTS\_TRANSACTIONS\n";

if (t[3] == '2')

tmp += "FILE\_SUPPORTS\_USN\_JOURNAL\n";

if (t[9] == '4')

tmp += "FILE\_UNICODE\_ON\_DISK\n";

if (t[6] == '8')

tmp += "FILE\_VOLUME\_IS\_COMPRESSED\n";

if (t[8] == '2')

tmp += "FILE\_VOLUME\_QUOTAS\n";

WriteFile(h, tmp.c\_str(), (tmp).size(), &size, &ovl);

size += (tmp).size();

}

else

{

string t = " Not Present (GetVolumeInformation)" + '\n';

WriteFile(h, t.c\_str(), (t).size(), &size, &ovl);

size += (t).size();

}

}

DWORD logicalDrivers(char \*path)

{

int n;

char dd[4];

string t = '\n'+"Available disk drives : ";

DWORD dr = GetLogicalDrives();

DWORD size;

HANDLE hEndRead; // дескриптор события

OVERLAPPED ovl; // структура управления асинхронным доступом к файлу

// создаем события с автоматическим сбросом

hEndRead = CreateEvent(NULL, FALSE, FALSE, NULL);

if (hEndRead == NULL)

return GetLastError();

// инициализируем структуру OVERLAPPED

ovl.Offset = 0xFFFFFFFF; // младшая часть смещения

ovl.OffsetHigh = 0xFFFFFFFF; // старшая часть смещения

ovl.hEvent = hEndRead;

HANDLE hFile = CreateFileW(ConvertToLPWSTR(path), GENERIC\_WRITE,

FILE\_SHARE\_WRITE, NULL, CREATE\_ALWAYS,

FILE\_FLAG\_OVERLAPPED, NULL);

if (hFile == INVALID\_HANDLE\_VALUE)

{

cerr << "Create file failed." << endl

<< "The last error code: " << GetLastError() << endl;

CloseHandle(hEndRead);

return 0;

}

for (int i = 0; i < 26; i++)

{

n = ((dr >> i) & 0x00000001);

if (n == 1)

{

dd[0] = char(65 + i); dd[1] = ':'; dd[2] = '\\'; dd[3] = 0;

t = "\nAvailable disk drives : " + (string)dd + '\n';

WriteFile(hFile, t.c\_str(), (t).size(), &size, &ovl);

size += (t).size();

volumeInformation(i, hFile, size, ovl);

driveType(dd, hFile, size, ovl);

}

}

}

void driveType(string s, HANDLE h, DWORD &size, OVERLAPPED ovl)

{

int d;

LPWSTR x = ConvertToLPWSTR(s);

//d = GetDriveType(L"c:\\");

d = GetDriveType(x);

string t;

if (d == DRIVE\_UNKNOWN) t = "\n Тип устройства UNKNOWN" + '\n';

if (d == DRIVE\_NO\_ROOT\_DIR) t = "\n Тип устройства DRIVE NO ROOT DIR" + '\n';

if (d == DRIVE\_REMOVABLE) t = "\n Тип устройства REMOVABLE" + '\n';

if (d == DRIVE\_FIXED) t = "\n Тип устройства FIXED" + '\n';

if (d == DRIVE\_REMOTE) t = "\n Тип устройства REMOTE" + '\n';

if (d == DRIVE\_CDROM) t = "\n Тип устройства CDROM" + '\n';

if (d == DRIVE\_RAMDISK) t = "\n Тип устройства RAMDISK" + '\n';

WriteFile(h, t.c\_str(), (t).size(), &size, &ovl);

size += (t).size();

}

LPWSTR ConvertToLPWSTR(const std::string& s)

{

LPWSTR ws = new wchar\_t[s.size() + 1]; // +1 for zero at the end

copy(s.begin(), s.end(), ws);

ws[s.size()] = 0; // zero at the end

return ws;

}

int createKey(char \*ret)

{

// Строка которую будем писать в реестр

\_TCHAR szTestString[] = \_T("

// Ключ который будем создавать

\_TCHAR szPath[] = \_T("Software\\");

HKEY hKey;

// Создаем ключ в ветке HKEY\_CURRENT\_USER

if (RegCreateKeyEx(HKEY\_CURRENT\_USER, szPath, 0, NULL, REG\_OPTION\_VOLATILE, KEY\_WRITE, NULL, &hKey, NULL) != ERROR\_SUCCESS) {

\_tcout << \_T("При создании ключа произошла ошибка") << endl;

return 1;

}

// Пишем строку в созданный ключ

if (RegSetValueEx(hKey, \_T("OS5"), 0, REG\_SZ, (BYTE\*)szTestString, sizeof(szTestString)) != ERROR\_SUCCESS) {

\_tcout << \_T("При записи строки произошла ошибка") << endl;

return 2;

}

// Закрываем хэндл ключа

if (RegCloseKey(hKey) != ERROR\_SUCCESS) {

\_tcout << \_T("При закрытии ключа произошла ошибка") << endl;

return 3;

};

\_TCHAR szBuf[MAX\_PATH];

DWORD dwBufLen = MAX\_PATH;

if (RegGetValue(HKEY\_CURRENT\_USER, szPath, \_T("OS5"), RRF\_RT\_REG\_SZ, NULL, (BYTE\*)szBuf, &dwBufLen) != ERROR\_SUCCESS) {

\_tcout << \_T("При чтении строки произошла ошибка") << endl;

return 4;

}

for (int i = 1; i < MAX\_PATH; i++)

ret[i-1] = szBuf[i];

cout << "Считанная строка " << ret << endl;

\_tsystem(\_T("pause"));

return 0;

}

string toHex(DWORD a)

{

string tmp("");

char t[9];

do

{

int r(a % 16);

if (r > 9) { r += (int)'A' - 10; }

else { r += (int)'0'; };

tmp = (char)r + tmp;

a /= 16;

} while (a);

int size = tmp.size();

int i;

t[8] = 0;

size--;

for (i = 7; size >= 0; i--)

t[i] = tmp[size--];

for (; i >= 0; i--)

t[i] = '0';

return t;

}





