Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Системное программное обеспечение вычислительных машин (СПОВМ)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

«Утилита проверки целостности файловой системы»

Студент: гр.350501 Толкачёв В.И.

Руководитель: Яночкин А.Л.

Минск 2015

Содержание

[Введение 3](#_Toc414805627)

[1.Обзор литературы 4](#_Toc414805628)

[2. Техническое задание 7](#_Toc414805629)

[3. Системное проектирование 8](#_Toc414805629)

[4. Функциональное проектирование 9](#_Toc414805629)

[5. Тестирование программы 12](#_Toc414805629)

[6. Руководство пользователя 14](#_Toc414805629)

[Заключение 18](#_Toc414805629)

Список литературы  [19](#_Toc414805630)

Приложение А  [2](#_Toc414805630)0

# Введение

Всем компьютерным приложениям нужно хранить и получать информацию. Во время работы процесс может хранить ограниченное количество данных в собственном адресном пространстве. Однако ёмкость такого хранилища ограничена размерами виртуального адресного пространства. Для некоторых приложений такого размера вполне достаточно, но для других, одного только виртуального базового пространства будет недостаточно.

Кроме того , после завершения работы процесса информация, хранящаяся в его адресном пространстве, теряется. Для большинства приложений эта информация должна храниться неделями, месяцами, или даже вечно. Исчезновение данных после завершения работы процесса для таких приложений неприемлемо. Информация должна сохраняться даже при аварийном завершении процесса в случае сбоя компьютера. Также проблема состоит в том, что часто возникает необходимость нескольким процессам одновременно получить доступ к одним и тем же данным. Необходимо отделить информацию от процесса.

Обычное решение всех проблем состоит в хранении информации на дисках и других внешних хранителях и модулях, называемых файлами. Информация , хранящаяся в файлах, должна обладать устойчивостью, файл должен исчезать только тогда, когда его владелец даст команду удаления файла. Часть операционной системы, работающая с файлами , называется файловой системой.

Файловую систему можно считать целостной, если один блок данных принадлежит одному файлу, т.е. изменение одного файла не приводит к изменению другого файла. Иногда при проверке файловой системы в **Windows** обнаруживалось, что один кластер принадлежит двум или более файлам одновременно.

В начале каждой файловой системы есть чистый бит. При подключении файловой системы этот бит стирается. Это означает, что файловая система используется в данный момент, а при завершении работы этот бит заменяется обратно в чистый. Если при загрузке чистый бит, не установлен, то ОС запускает средство проверки файловой системы. В **Windows** это программа – **CHKDSK**, а в **Linux** программа ***fsck***. Программы проверяют целостность файловой системы.

# 1.Обзор литературы

**CHKDSK**(от англ. checkdisk — "проверка диска"). Как уже писал, данное приложение служит для диагностики и исправления ошибок файловой системы. При необходимости можно проверить жесткий диск на поврежденные сектора, которые соответствующим образом метятся и в дальнейшей "жизни" Windows участия не принимают. Для всех вышеперечисленных манипуляций от Вас требуются права администратора.

CHKDSK запускается в автоматическом режиме после серьезных сбоев в работе системы и это хорошее напоминание для нерадивого пользователя сделать бэкап (от англ. backup – "резервная копия") ценной информации и подумать о возможной замене жесткого диска. По умолчанию, CHKDSK только диагностирует файловую систему на наличие ошибок, а исправлять их и проверять жесткий диск Вам придется в "ручном" режиме.

В целях профилактики подобных ситуаций рекомендуется периодически самостоятельно активировать данное стандартное приложение Windows.

CHKDSK сканирует всю поверхность диска на возможные ошибки распределения и генерирует отчет, где указывается размер оставшегося пространства (в байтах), объем используемого пространства и число существующих файлов (включая скрытые и ожидающие удаления файлы), объем доступной памяти и найденные ошибки.

Параметр "диск:" задает дисковод с проверяемым диском. Параметр "спецификация\_файлов" задает расположение и имя файла или набора файлов (при указании трафаретных символов \* и ?), фрагментацию которых вы хотите проверить с помощью CHKDSK.

Если вы задаете в CHKDSK спецификацию файла, CHKDSK проверяет, что каждый файл состоит из смежных кластеров на диске и выводит имен файлов, состоящих из несмежных кластеров. Если вы укажете параметр /F, то CHKDSK будет корректировать найденные ошибки. Файлы, ссылающиеся на один и тот же блок, не корректируются даже при указании параметра /F (о них только сообщается). Чтобы исправить их, скопируйте данные файлы в другие каталоги и удалите оригиналы. Часть информации в этих файлах может оказаться потерянной.

Этапы работы CHKDSK:

Этап 1 .Проверка файлов  
Во время первого прохода CHKDSK выводит сообщение о том, что выполняется проверка файлов, а также объем выполненной проверки, выраженный в процентах (от 0 до 100). В течение этого этапа CHKDSK проверяет сегмент записи каждого файла в основной таблице файлов тома.  
  
Этап 2. Проверка индексов  
  
По существу, индексы - это каталоги файловой системы. CHKDSK выполняет проверку того, что нет "потерянных" файлов и что во всех списках каталогов содержатся существующие файлы. Потерянным называется файл, для которого существует правильный сегмент записи файла, но о котором нет данных ни в одном списке каталога. Потерянный файл может быть восстановлен в соответствующем ему каталоге, если этот каталог еще существует. Если соответствующий каталог более не существует, CHKDSK создаёт каталог в корневом каталоге диска и перемещает файл в него.   
  
Этап 3. Проверка дескрипторов безопасности  
  
В дескрипторах безопасности содержатся сведения о владельце файла или каталога, о разрешениях файловой системы для данного файла или каталога, и об аудите для данного файла или каталога. CHKDSK проверяет структуру каждого дескриптора безопасности, но не выполняет проверку реального существования перечисленных пользователей или групп и правомерность предоставленных разрешений.   
  
Этап 4. Проверка секторов  
  
Данный этап выполнения CHKDSK определяется наличием параметра /R при запуске программы. Выполняется поиск поврежденных секторов в свободном пространстве тома. CHKDSK выполняет попытку чтения каждого сектора на томе, и , при обнаружении ошибки, кластер, в который входит данный сектор, помечается как дефектный и исключается из логической структуры тома. Даже без использования ключа /R программа всегда проверяет чтением секторы, относящиеся к таблице MFT (к метаданным ). Кроме того, секторы, которые используются для области пользовательских данных, проверяются на предыдущих этапах работы CHKDSK.   
  
Необходимо учитывать тот факт, что время выполнения CHKDSK с ключом /R может быть значительным. Кроме того, современные жесткие диски имеют встроенную систему самотестирования и контроля параметров (S.M.A.R.T) , наличие которой делает бессмысленным использование режима поиска поврежденных секторов с помощью CHKDSK, поскольку все современные накопители постоянно выполняют внутренние подпрограммы контроля технического состояния и самодиагностики, а также встроенные на микропрограммном уровне процедуры переназначения плохо читающихся секторов ( нестабильных секторов ) на секторы из резервной области ( процедура remap или ремап ). Данные процессы происходят невидимо для пользователя компьютера. Поэтому, наличие сбойных блоков ( Bad Blocks ) возможно только при отсутствии свободного места в резервной области для переназначения, или при возникновении сбоев в момент записи данных в сектор, например, при аварийном выключении первичного электропитания.   
  
При обнаружении потерянных файлов, программа CHKDSK создает их в структуре файловой системы в виде файлов с расширением .CHK. На практике, информация из таких файлов может быть восстановлена вручную только в тех случаях, когда она представлена в текстовом виде. В некоторых случаях, можно воспользоваться программным обеспечением сторонних производителей

**2. Техническое задание**

* 1. Общие сведения

Программное средство, реализуемое в данном курсовом проекте, представляет из себя утилиту, позволяющую пользователю проверить целостность файловой системы. Используемые языки программирования – С, С++.

* 1. Назначение и цели создания продукта

Программа позволит проверить жёсткий диск  на ошибки файловой системы, а также может исправить найденные ошибки файловой системы, может проверять диски на наличие физически повреждённых секторов.

* 1. Требования к программному продукту

Требуется разработать программу, проверяющую целостность файловой системы.

Необходимо предусмотреть возможности:

- исправление ошибок на диске

- нахождение повреждённых секторов и восстановление информации

-  диагностика диска

**3.** Системное проектирование

Данный курсовой проект включает в себя несколько важных блоков , описанных в данном разделе.

## 3.1 Способ взаимодействия пользователя и программы.

Для обеспечения взаимодействия с пользователем был выбран консольный интерфейс.

Рассмотрим взаимодействие программы с пользователем. Пользователю необходимо запустить приложение и ввести в командной строке ключ для выполнения определённой функции программы. После этого программа будет выполнять анализ файловой системы.

## 3.2 Структурная схема программы

Работу каждой программы можно разделить на блоки, которые будут описаны далее.

- Основной логический модуль.

-Динамическая библиотека Fmifs.dll(Format Manager for Installable File Systems), . Данная динамическая библиотека предоставляет интерфейсы для запуска основного логического модуля.

-Блок проверки файлов.

-Блок проверки индексов.

-Блок проверки дескрипторов безопасности.

Последние 3 блока выполняют основную диагностику файловой системы.

Fmifs.dll

Основной логический модуль

Блок проверки дескрипторов безопасности.

Блок проверки индексов.

Блок проверки файлов.

Рис. 3.2 – Структурная схема программы

**4.** Функциональное проектирование

VOID Usage(PWCHAR ProgramName)— Памятка , описывающая пользователю , как использовать данную программу. Данная памятка , включает в себя команды , которые необходимо ввести в командную строку(исправление ошибок на диске, нахождение повреждённых секторов и восстановление информации, диагностика диска ,если он повреждён).

Листинг метода VOID Usage

VOID Usage(PWCHAR ProgramName)

{

\_tprintf(L"Usage: %s [drive:] [-R] [-C]\n\n"); \_

tprintf(L" [drive:] Specifies the drive to check.\n");

\_tprintf(L" -R Locates bad sectors and recovers readable information.\n"); \_tprintf(L" -C Checks the drive only if it is dirty.\n"); \_tprintf(L"\n");

}

int ParseCommandLine(int argc, WCHAR \* argv[]) — данная функция включает в себя работу с ключами , которые мы вводим в командную строку для работы с приложением.

Листинг метода int ParseCommandLine

int ParseCommandLine( int argc, WCHAR \*argv[] )

{

int i;

BOOLEAN gotFix = FALSE;

BOOLEAN gotClean = FALSE;

for( i = 1; i < argc; i++ ) {

switch( argv[i][0] ) {

case '-':

case '/':

switch( argv[i][1] ) {

case L'R':

case L'r':

if( gotFix ) return i;

ScanSectors = TRUE;

gotFix = TRUE;

break;

case L'C':

case L'c':

if( gotClean ) return i;

SkipClean = TRUE;

gotClean = TRUE;

break;

default:

return i;

}

break;

default:

if( Drive ) return i;

if( argv[i][1] != L':' ) return i;

Drive = argv[i];

break;

}

}

return 0;

}

void PrintWin32Error(PWCHAR Message, DWORD ErrorCode)—Нахождение ошибки кода , и вывод сообщения ошибки.

Листинг метода void PrintWin32Error

void PrintWin32Error( PWCHAR Message, DWORD ErrorCode )

{

LPVOID lpMsgBuf;

FormatMessageW( FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM,

NULL, ErrorCode,

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT),

(PWCHAR) &lpMsgBuf, 0, NULL );

\_tprintf(L"%s: %s\n", Message, lpMsgBuf );

LocalFree( lpMsgBuf );

}

BOOLEAN \_\_stdcall ChkdskCallback(CALLBACKCOMMAND Command, PVOID Argument)— Библиотека файловой системы будет связывать нас с командами, которые мы можем использовать. В CALLBACKCOMMAND Command мы передаём команду , с которой будет работать данная программа , а в PVOID Argument передаётся состояние выполнения данной команды.

Листинг метода BOOLEAN \_\_stdcall ChkdskCallback

BOOLEAN \_\_stdcall ChkdskCallback( CALLBACKCOMMAND Command, PVOID Argument )

{

PDWORD percent;

PBOOLEAN status;

PTEXTOUTPUT output;

switch( Command ) {

case PROGRESS:

percent = (PDWORD) Argument;

\_tprintf(L"%d percent completed.\r", \*percent);

break;

case OUTPUT:

output = (PTEXTOUTPUT) Argument;

fprintf(stdout, "%s", output->Output);

break;

case DONE:

status = (PBOOLEAN) Argument;

if( \*status == TRUE ) {

\_tprintf(L"Chkdsk was unable to complete successfully.\n\n");

Error = TRUE;

}

break;

}

return TRUE;

}

BOOLEAN LoadFMIFSEntryPoints()—загружает FMIFS.DLL и находит точку входа , которую мы собираемся использовать.

Листинг метода BOOLEAN LoadFMIFSEntryPoints

BOOLEAN LoadFMIFSEntryPoints()

{

LoadLibrary( "fmifs.dll" );

if( !(Chkdsk = (void \*) GetProcAddress( GetModuleHandle( "fmifs.dll"),

"Chkdsk" )) ) {

return FALSE;

}

return TRUE;

}

BOOL WINAPI GetVolumeInformationW ( Drive, volumeName, sizeof(volumeName), &serialNumber, &maxComponent, &flags, fileSystem, sizeof(fileSystem))) — функция получает информацию о файловой системе и объема, связанные с указанным корневом каталоге.Где Drive —указатель на строку , которая содержит корневой каталог тома ,volumeName — указатель на буфер, который получает имя указанного тома , sizeof(volumeName)— размер имени тома буфера , serialNumber— кказатель на переменную, которая получает серийный номер тома, maxComponent— имя файлового компонента, который поддерживает указанная файловая система, flags— указатель на переменную, которая получает флаги, связанные с указанной файловой системы, fileSystem —указатель на буфер, который получает имя файловой системы, sizeof(fileSystem)— длина буфера имени файловой системы.

**5. ТЕСТИРОВАНИЕ ПРОГРАММЫ**

Для тестирования программы использовалось стандартное приложение CHKDSK от Microsoft.

Для запуска программы следует вызвать командную строку и ввести в ней название программы и ключ:

/F — выполнение проверки на наличие ошибок и их автоматическое исправление;

/V — в процессе проверки диска выводить полные пути и имена хранящихся на диске файлов. Для дисков, содержащих разделы NTFS, также выводятся сообщения об очистке;

/R — выполнить поиск поврежденных секторов и восстановить их содержимое. Включает в себя действие ключа /F;

Выведем полные пути и имена хранящихся на диске файлов. Для этого в командной строке введём команду :

c:\Users\user>chkdsk /V

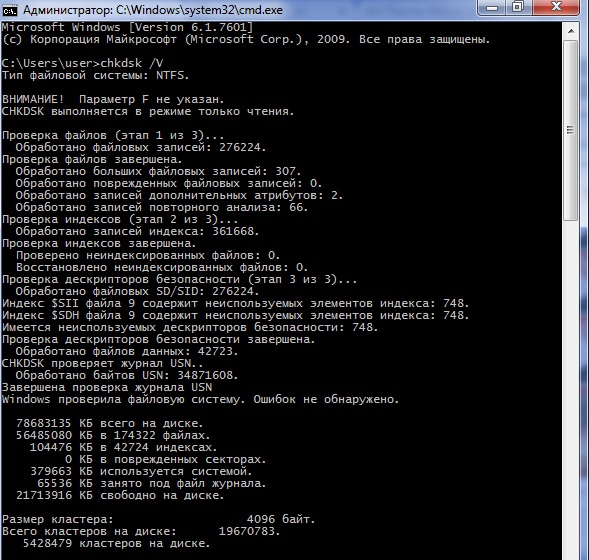


Рис. 5.1

Для выполнения проверки на наличие ошибок и их автоматическое исправление в командной строке введём следующую команду :

c:\Users\user>chkdsk /F

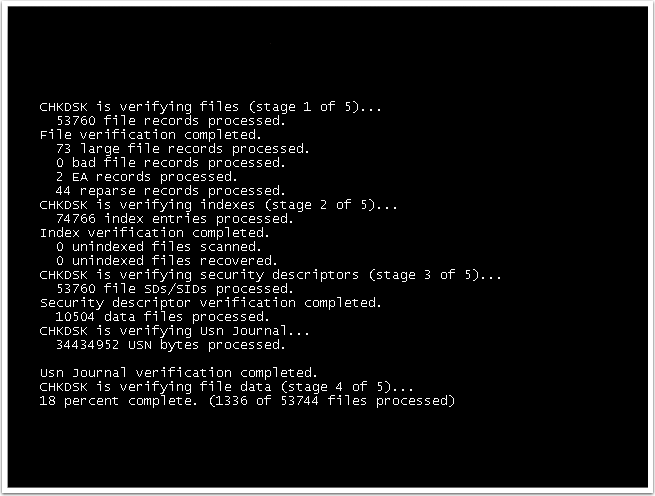


Рис. 5.2

**6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Для тестирования программы использовалась программа Microsoft Visual Studio 2013:

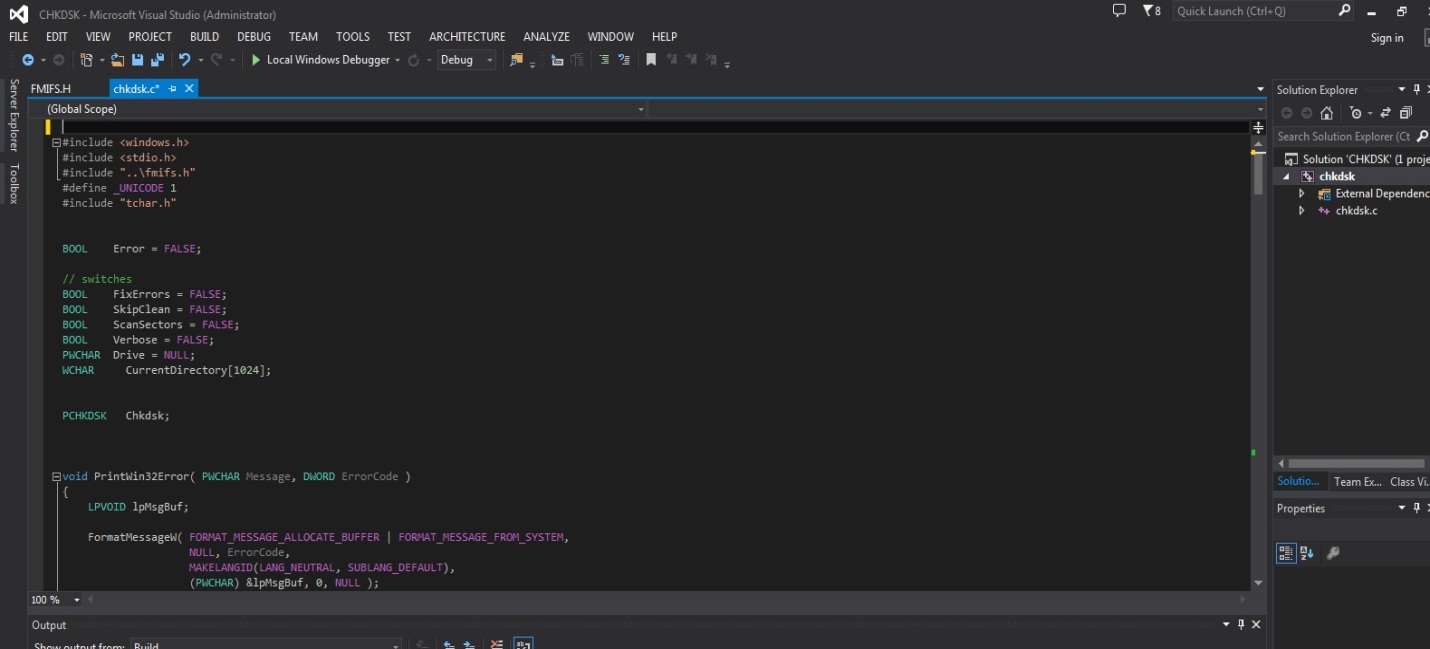
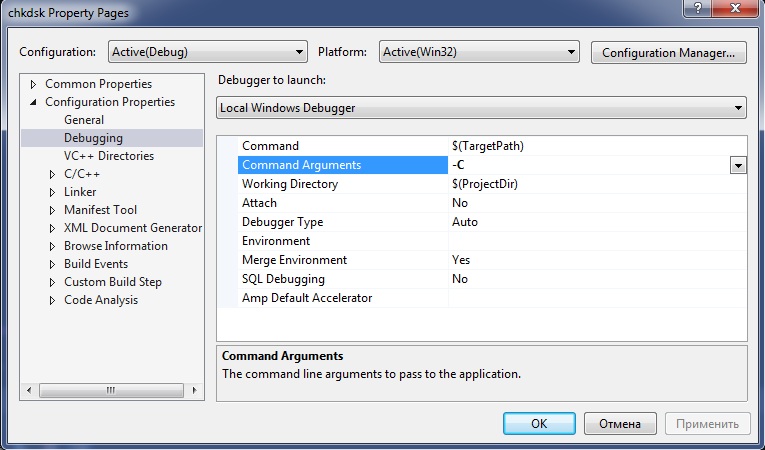


Рис. 6.1

Для работы с данной программой необходимо в командной строке ввести ключ , для того чтобы программа выполняла определённую функцию.

Для проверки диска ,если он содержит файлы ,содержащие ошибки , необходимо ввести ключ “-C”.

 Рис. 6.2

Как показало тестирование , ошибок на диске не обнаружено.

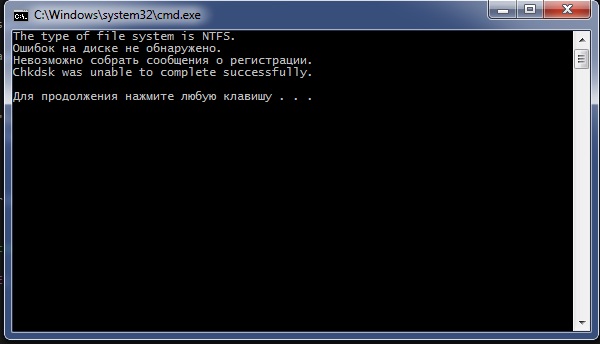


Рис. 6.3

Для полной диагностики , нахождения плохих секторов и восстановления той части данных, мы используем ключ “-R”

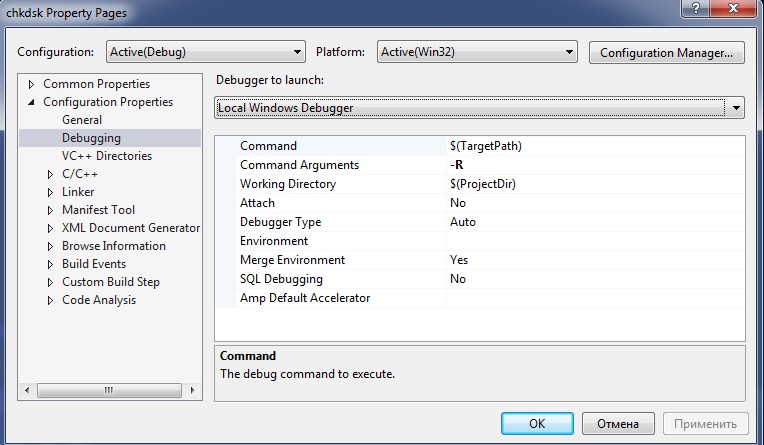


Рис. 6.4

Результат работы показал ,что файловая система ошибок не содержит. Следует заметить , что пункт, который проверяет свободное пространство на диске занимает очень много времени(около 50 минут).

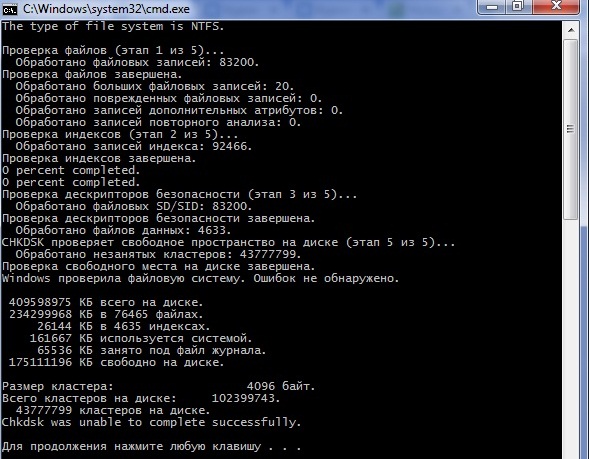


Рис. 6.5

Если в командную строку не вводить никаких ключей , то программа выполнит диагностику файловой системы в три этапа:

-этап проверки файлов

-этап проверки индексов

-этап проверки дескрипторов безопасности

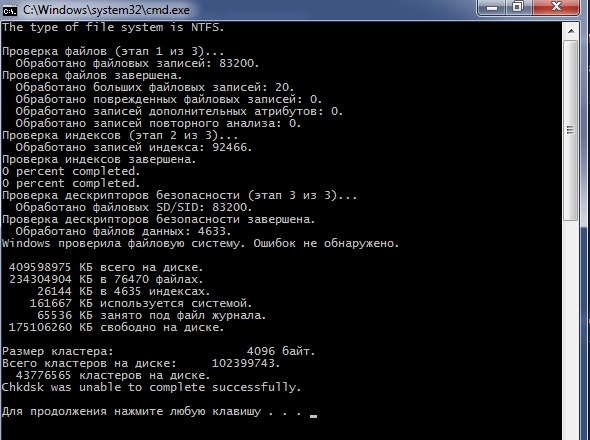


Рис. 6.6

Заключение

В ходе выполнения курсового проекта были спроектирована и разработана утилита проверки целостности файловой системы. Данная программа диагностирует и исправляет ошибки файловой системы. В будущем планируется разработать приложение для более глубокого анализа файловой системы на различных операционных системах. В ходе разработки данного проекта, были получены обширные знания в разработке системного программного обеспечения для Windows.

Список литературы

# 1. Э. Таненбаум. Современные операционные системы, 3-е издание : Издательство: Питер, 2010. – 1116 с.

**2.** CHKDSK — Википедия [Электронный ресурс]. – Электронные данные. –

<https://ru.wikipedia.org/wiki/CHKDSK>.

**3.** GetVolumeInformation function (Windows) - MSDN - Microsoft

[Электронный ресурс]. – Электронные данные. –

<https://msdn.microsoft.com/enus/library/windows/desktop/aa364993%28v=vs.85%29.aspx>

**4.**\_\_stdcall - MSDN - Microsoft [Электронный ресурс]. – Электронные данные. –<https://msdn.microsoft.com/en-us/library/zxk0tw93.aspx>

ПРИЛОЖЕНИЕ А

Листинг кода файла chkdsk.c:

#include <windows.h>

#include <stdio.h>

#include "..\fmifs.h"

#define \_UNICODE 1

#include "tchar.h"

VOID Usage(PWCHAR ProgramName)

{

\_tprintf(L"Использование: %s [drive:] [-R] [-C]\n\n");

\_tprintf(L" [drive:] Определяет диск, чтобы проверить.\n");

;

\_tprintf(L" -R Нахождение плохих секторов и восстановление той части данных.\n");

\_tprintf(L" -C Проверяет диск, только если он загрязнен.\n");

\_tprintf(L"\n");

}

void PrintWin32Error(PWCHAR Message, DWORD ErrorCode)

{

LPVOID lpMsgBuf;

FormatMessageW(FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM,

NULL, ErrorCode,

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT),

(PWCHAR)&lpMsgBuf, 0, NULL);

\_tprintf(L"%s: %s\n", Message, lpMsgBuf);

LocalFree(lpMsgBuf);

}

int ParseCommandLine(int argc, WCHAR \*argv[])

{

int i;

BOOLEAN gotFix = FALSE;

BOOLEAN gotClean = FALSE;

for (i = 1; i < argc; i++) {

switch (argv[i][0]) {

case '-':

case '/':

switch (argv[i][1]) {

case L'R':

case L'r':

if (gotFix) return i;

ScanSectors = TRUE;

gotFix = TRUE;

break;

case L'C':

case L'c':

if (gotClean) return i;

SkipClean = TRUE;

gotClean = TRUE;

break;

default:

return i;

}

break;

default:

if (Drive) return i;

if (argv[i][1] != L':') return i;

Drive = argv[i];

break;

}

}

return 0;

}

//------------------------------------------------------------

BOOLEAN \_\_stdcall ChkdskCallback(CALLBACKCOMMAND Command, PVOID Argument)

{

PDWORD percent;

PBOOLEAN status;

PTEXTOUTPUT output;

switch (Command) {

case PROGRESS:

percent = (PDWORD)Argument;

\_tprintf(L"%d percent completed.\r", \*percent);

break;

case OUTPUT:

output = (PTEXTOUTPUT)Argument;

fprintf(stdout, "%s", output->Output);

break;

case DONE:

status = (PBOOLEAN)Argument;

if (\*status == TRUE) {

\_tprintf(L"Chkdsk was unable to complete successfully.\n\n");

Error = TRUE;

}

break;

}

return TRUE;

}

BOOLEAN LoadFMIFSEntryPoints()

{

LoadLibrary("fmifs.dll");

if (!(Chkdsk = (void \*)GetProcAddress(GetModuleHandle("fmifs.dll"),

"Chkdsk"))) {

return FALSE;

}

return TRUE;

}

//---------------------------------------------------------------------

int wmain(int argc, WCHAR \*argv[])

{

int badArg;

HANDLE volumeHandle;

WCHAR fileSystem[1024];

WCHAR volumeName[1024];

DWORD serialNumber;

DWORD flags, maxComponent;

if (!LoadFMIFSEntryPoints()) {

\_tprintf(L"Could not located FMIFS entry points.\n\n");

return -1;

}

if ((badArg = ParseCommandLine(argc, argv))) {

\_tprintf(L"Unknown argument: %s\n", argv[badArg]);

return -1;

}

if (!Drive) {

if (!GetCurrentDirectoryW(sizeof(CurrentDirectory), CurrentDirectory)) {//Retrieves the current directory for the current process

PrintWin32Error(L"Could not get current directory", GetLastError());

return -1;

}

}

else {

wcscpy(CurrentDirectory, Drive);

}

CurrentDirectory[2] = L'\\';

CurrentDirectory[3] = (WCHAR)0;

Drive = CurrentDirectory;

if (!GetVolumeInformationW(Drive,

volumeName, sizeof(volumeName),

&serialNumber, &maxComponent, &flags,

fileSystem, sizeof(fileSystem))) {

PrintWin32Error(L"Could not query volume", GetLastError());

return -1;

}

if (FixErrors) {

swprintf(volumeName, L"\\\\.\\%C:", Drive[0]);

volumeHandle = CreateFileW(volumeName, GENERIC\_WRITE,

0, NULL, OPEN\_EXISTING,

0, 0);

if (volumeHandle == INVALID\_HANDLE\_VALUE) {

\_tprintf(L"Chdskx cannot run because the volume is in use by another process.\n\n");

return -1;

}

CloseHandle(volumeHandle);

SetConsoleCtrlHandler(CtrlCIntercept, TRUE);

}

\_tprintf(L"The type of file system is %s.\n", fileSystem);

Chkdsk(Drive, fileSystem, FixErrors, Verbose, SkipClean, ScanSectors,

NULL, NULL, ChkdskCallback);

if (Error) return -1;

return 0;

}