#include <Windows.h>

#include <iostream>

#include <io.h>

#include <fcntl.h>

using namespace std;

#define FILE\_PATH L"F:/.Уник/3 курс/ОС/9laba/lab09/lab09/OS09\_01.txt"

BOOL printFileInfo(LPWSTR FileName);

string getFileName(wchar\_t\* filePath);

LPCWSTR getFileType(HANDLE file);

BOOL printFileTxt(LPWSTR FileName);

int main()

{

setlocale(LC\_ALL, "Rus");

LPWSTR path = (LPWSTR)FILE\_PATH;

printFileInfo(path);

printFileTxt(path);

}

BOOL printFileInfo(LPWSTR path)

{

HANDLE file = CreateFile(

path,

GENERIC\_READ,

NULL,

NULL,

OPEN\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

SYSTEMTIME sysTime;

BY\_HANDLE\_FILE\_INFORMATION fi;

BOOL fResult = GetFileInformationByHandle(file, &fi);

if (fResult)

{

FileTimeToSystemTime(&fi.ftCreationTime, &sysTime);

cout << "Creation time: " << sysTime.wDay << '.' << sysTime.wMonth << '.' << sysTime.wYear << " " << sysTime.wHour + 3 << '.' << sysTime.wMinute << '.' << sysTime.wSecond;

FileTimeToSystemTime(&fi.ftLastWriteTime, &sysTime);

cout << "\nLast update time: " << sysTime.wDay << '.' << sysTime.wMonth << '.' << sysTime.wYear << " " << sysTime.wHour + 3 << '.' << sysTime.wMinute << '.' << sysTime.wSecond;

cout << "\nFile name: " << getFileName((wchar\_t\*)FILE\_PATH);

wcout << "\nFile type: " << getFileType(file);

cout << "\nFile size: " << fi.nFileSizeHigh << '.' << fi.nFileSizeLow;

}

CloseHandle(file);

return true;

}

BOOL printFileTxt(LPWSTR path)

{

HANDLE file = CreateFile(

path,

GENERIC\_READ,

NULL,

NULL,

OPEN\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

// Буфер для чтения файла

char buf[1024];

DWORD countBytes = 0;

ZeroMemory(buf, sizeof(buf));

// Чтение файла с использованием функции ReadFile

BOOL b = ReadFile(file, &buf, sizeof(buf) - 1, &countBytes, NULL);

if (!b) throw "read file failed";

int wideStrLen = MultiByteToWideChar(CP\_UTF8, 0, buf, -1, NULL, 0);

wchar\_t\* wideStr = new wchar\_t[wideStrLen];

int result = MultiByteToWideChar(CP\_UTF8, 0, buf, -1, wideStr, wideStrLen);

int ansiLen = WideCharToMultiByte(CP\_ACP, 0, wideStr, -1, NULL, 0, NULL, NULL);

char\* ansiStr = new char[ansiLen];

result = WideCharToMultiByte(CP\_ACP, 0, wideStr, -1, ansiStr, ansiLen, NULL, NULL);

cout << "\n-- Read file " << countBytes << " byte succesfull:\n" << ansiStr << '\n';

}

LPCWSTR getFileType(HANDLE file)

{

switch (GetFileType(file))

{

case FILE\_TYPE\_UNKNOWN:

return L"FILE\_TYPE\_UNKNOWN";

case FILE\_TYPE\_DISK:

return L"FILE\_TYPE\_DISK";

case FILE\_TYPE\_CHAR:

return L"FILE\_TYPE\_CHAR";

case FILE\_TYPE\_PIPE:

return L"FILE\_TYPE\_PIPE";

case FILE\_TYPE\_REMOTE:

return L"FILE\_TYPE\_REMOTE";

default:

return L"ERROR: WRITE FILE TYPE";

}

}

string getFileName(wchar\_t\* filePath)

{

wstring ws(filePath);

string filename(ws.begin(), ws.end());

const size\_t last\_slash\_idx = filename.find\_last\_of("\\/");

if (string::npos != last\_slash\_idx)

{

filename.erase(0, last\_slash\_idx + 1);

}

return filename;

}

//

//FileTimeToSystemTime - это функция в Windows API, используемая для преобразования времени в формате FILETIME в формат SYSTEMTIME.

//

//cpp

//Copy code

//BOOL FileTimeToSystemTime(

// const FILETIME\* lpFileTime, // указатель на структуру FILETIME

// LPSYSTEMTIME lpSystemTime // указатель на структуру SYSTEMTIME

//);

//HANDLE file = CreateFile(

// path, // LPCTSTR lpFileName: Путь к файлу

// GENERIC\_READ, // DWORD dwDesiredAccess: Запрашиваемый доступ к файлу

// NULL, // DWORD dwShareMode: Режим совместного доступа

// NULL, // LPSECURITY\_ATTRIBUTES lpSecurityAttributes: Атрибуты безопасности (в данном случае, отсутствуют)

// OPEN\_ALWAYS, // DWORD dwCreationDisposition: Действие при создании файла (открывать существующий, если существует) если файл существует, он должен быть открыт, а если не существует, то создан.Другие возможные значения включают CREATE\_NEW, CREATE\_ALWAYS, и т.д.

// FILE\_ATTRIBUTE\_NORMAL, // DWORD dwFlagsAndAttributes: Атрибуты файла (обычный файл)

// NULL); // HANDLE hTemplateFile: Шаблон файла (не используется, установите в NULL)

//

//FILE\_TYPE\_UNKNOWN: Этот тип файла указывает, что тип файла неизвестен.Вероятно, это может произойти, если система не может определить тип файла.

//

//FILE\_TYPE\_DISK : Этот тип файла указывает на дисковый файл.Такие файлы обычно хранятся на физических носителях данных, таких как жесткие диски.

//

//FILE\_TYPE\_CHAR : Этот тип файла указывает на символьное устройство.Символьные устройства предоставляют поток символов, и их можно читать или записывать как последовательность символов.

//

//FILE\_TYPE\_PIPE : Этот тип файла указывает на канал(pipe).Каналы - это механизм межпроцессного взаимодействия, где данные могут передаваться между процессами.

//

//FILE\_TYPE\_REMOTE : Этот тип файла указывает на удаленный файл.Такие файлы находятся на удаленном устройстве или сервере.

//

//default: Если встречается тип файла, который не соответствует ни одному из указанных в case, возвращается сообщение об ошибке "ERROR: WRITE FILE TYPE".

//

//wstring ws(filePath); : Здесь создается объект wstring(строка широких символов) из переданного wchar\_t\* пути к файлу.Это нужно для работы с путями, содержащими символы широких символов(Unicode).

// если слеш найден то удаляем строку от начала до позиции слеша включая

//if (string::npos != last\_slash\_idx)

//{

// filename.erase(0, last\_slash\_idx + 1);

//}

//BOOL b = ReadFile(file, &buf, sizeof(buf) - 1, &countBytes, NULL);

//HANDLE file : Дескриптор файла, из которого будут считываться данные.

//

//LPVOID lpBuffer : Указатель на буфер, в который будут записываться считанные данные.

//

//DWORD nNumberOfBytesToRead : Количество байтов, которое нужно прочитать из файла.

//

//LPDWORD lpNumberOfBytesRead : Указатель на переменную, в которую будет записано реальное количество считанных байт.

//

//LPOVERLAPPED lpOverlapped : Указатель на структуру OVERLAPPED, используемую для асинхронного ввода - вывода.В данном случае, так как последний параметр установлен в NULL, операция чтения будет синхронной.

//int wideStrLen = MultiByteToWideChar(CP\_UTF8, 0, buf, -1, NULL, 0); из UTF8 в широкую строку (Unicode).

//UINT CodePage : Кодовая страница, используемая для преобразования.В данном случае, CP\_UTF8, что означает, что входные данные представляют собой UTF - 8 кодированный текст.

//

//DWORD dwFlags : Флаги для дополнительных опций конвертации.В данном случае, параметр установлен в 0, что указывает на отсутствие дополнительных опций.

//

//LPCCH lpMultiByteStr : Указатель на строку, содержащую последовательность байтов(в данном случае, ваш буфер buf).

//

//int cbMultiByte : Количество байт во входной строке.В данном случае, -1 означает, что входная строка является нуль - терминированной, и длина будет автоматически определена.

//

//LPWSTR lpWideCharStr : Указатель на буфер, в который будут записаны широкие символы.

//

//int cchWideChar : Максимальное количество широких символов, которые могут быть записаны в буфер.Если этот параметр установлен в 0, функция вычислит необходимый размер буфера.

//

//широкой строки в ANSI - кодировку.

//

//CP\_ACP: Это константа, представляющая кодовую страницу системы(ANSI code page).

//

//0 : Флаги.В данном случае, они установлены в 0, что означает использование стандартных опций преобразования.

//

//wideStr : Указатель на входную широкую строку(Unicode).

//

//- 1 : Количество символов входной строки для преобразования.Значение - 1 указывает на то, что строка должна быть преобразована полностью до завершающего символа NULL.

//

//NULL : Указатель на буфер для хранения многобайтовой строки.Если этот параметр установлен в NULL, функция возвращает размер буфера, не записывая данные.

//

//0 : Размер выходного буфера в байтах.

//

//NULL : Указатель на переменную, которая будет содержать количество байтов, записанных в буфер.Если параметр равен NULL, эта информация не возвращается.

//

//NULL : Указатель на переменную, которая будет содержать значение замещающего символа(если он используется).В данном случае, параметр не используется, поэтому передается NULL.

**2.---------------------------------------------------------------------------------**

#include <Windows.h>

#include <iostream>

using namespace std;

#define FILE\_PATH L"F:/.Уник/3 курс/ОС/9laba/lab09/lab09/OS09\_01.txt"

BOOL printFileTxt(LPWSTR FileName);

BOOL delRowFileTxt(LPWSTR FileName, DWORD row);

int main()

{

setlocale(LC\_CTYPE, "Rus");

LPWSTR path = (LPWSTR)FILE\_PATH;

delRowFileTxt(path, 1);

delRowFileTxt(path, 3);

delRowFileTxt(path, 8);

delRowFileTxt(path, 10);

printFileTxt(path);

}

BOOL printFileTxt(LPWSTR path)

{

HANDLE file = CreateFile(

path,

GENERIC\_READ,

NULL,

NULL,

OPEN\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

char buf[1024];

DWORD countBytes = 0;

ZeroMemory(buf, sizeof(buf));

BOOL b = ReadFile(file, &buf, sizeof(buf) - 1, &countBytes, NULL);

if (!b) throw "read file failed";

int wideStrLen = MultiByteToWideChar(CP\_UTF8, 0, buf, -1, NULL, 0);

wchar\_t\* wideStr = new wchar\_t[wideStrLen];

int result = MultiByteToWideChar(CP\_UTF8, 0, buf, -1, wideStr, wideStrLen);

int ansiLen = WideCharToMultiByte(CP\_ACP, 0, wideStr, -1, NULL, 0, NULL, NULL);

char\* ansiStr = new char[ansiLen];

result = WideCharToMultiByte(CP\_ACP, 0, wideStr, -1, ansiStr, ansiLen, NULL, NULL);

cout << "\n-- Read file " << countBytes << " byte succesfull:\n" << ansiStr << '\n';

}

BOOL delRowFileTxt(LPWSTR FileName, DWORD row) {

PLARGE\_INTEGER fileSize = new LARGE\_INTEGER();

int rowCount = 1;

int position = 0;

int positionAfter = 0;

bool rowFound = false;

try {

if (row <= 0) {

throw "Invalid number of raw";

}

HANDLE of = CreateFile(

FileName,

GENERIC\_READ | GENERIC\_WRITE,

FILE\_SHARE\_READ,

NULL,

OPEN\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

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

CloseHandle(of);

throw "Open file failed";

}

if (GetFileSizeEx(of, fileSize)) {

char\* buf = new char[fileSize->QuadPart + 1];

char\* bufAfterDel = new char[fileSize->QuadPart + 1];

ZeroMemory(buf, fileSize->QuadPart);

ZeroMemory(bufAfterDel, fileSize->QuadPart);

DWORD n = NULL;

if (ReadFile(of, buf, fileSize->QuadPart, &n, NULL)) {

buf[fileSize->QuadPart] = '\0';

//удаление строки из буфера

while (buf[position] != '\0') {

if (buf[position] == '\n') {

rowCount++;

}

if (rowCount == row) {

rowFound = true;

rowCount++;

position++;

while (buf[position] != '\n' && buf[position] != '\0') {

position++;

}

if (row == 1) {

position++;

}

}

if (position < fileSize->QuadPart) {

bufAfterDel[positionAfter++] = buf[position++];

}

else if(row == 1) {

bufAfterDel[0] = '\0';

break;

}

if (buf[position] == '\0') {

break;

}

}

bufAfterDel[positionAfter] = '\0';

if (bufAfterDel[positionAfter] == '\0' && bufAfterDel[positionAfter - 1] == '\r') {

bufAfterDel[positionAfter - 1] = '\0';

}

else {

bufAfterDel[positionAfter] = '\0';

}

}

if (rowFound) {

if (SetFilePointer(of, 0, 0, FILE\_BEGIN) == 0) {

if (WriteFile(of, bufAfterDel, strlen(bufAfterDel), &n, NULL)) {

std::cout << "Row deleted successfully #" << row << std::endl;

}

else {

CloseHandle(of);

throw "WriteFile failed";

}

if (!SetEndOfFile(of)) {

CloseHandle(of);

throw "SetEndOfFile failed";

}

}

else {

CloseHandle(of);

throw "SetFilePointer failed";

}

}

else {

CloseHandle(of);

throw "Row isn't found";

}

}

else {

CloseHandle(of);

throw "GetFileSizeEx failed";

}

CloseHandle(of);

}

catch (const char\* err) {

std::cout << "--- Error:\n" << err << "\n";

return false;

}

return true;

}

//

//Функция GetFileSizeEx используется для получения размера файла.Вот параметры этой функции :

//

//of: Дескриптор файла(HANDLE), который был открыт с использованием функции CreateFile.

//

//fileSize : Указатель на структуру LARGE\_INTEGER, куда будет записан размер файла.

//

//Функция SetFilePointer устанавливает текущую позицию в файле.Вот параметры этой функции :

//

//of: Дескриптор файла(HANDLE), который был открыт с использованием функции CreateFile.

//

//lDistanceToMove : Относительное смещение, которое будет добавлено или вычтено из текущей позиции в файле.Этот параметр может быть положительным, отрицательным или нулевым.

//

//lpDistanceToMoveHigh : Указатель на переменную, в которую будет записано значение старших битов lDistanceToMove.Если этот параметр равен NULL, старшие биты игнорируются.

//

//dwMoveMethod : Способ перемещения, который определяет, как интерпретировать lDistanceToMove.Это может быть одно из следующих значений :

//

//FILE\_BEGIN: Начало файла.lDistanceToMove трактуется как относительное смещение от начала файла.

//FILE\_CURRENT : Текущая позиция.lDistanceToMove трактуется как относительное смещение от текущей позиции в файле.

//FILE\_END : Конец файла.lDistanceToMove трактуется как относительное смещение от конца файла.

**3-----------------------------------------------------------------------------------------**

#define \_CRT\_NON\_CONFORMING\_WCSTOK

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cstdlib>

#include "Windows.h"

using namespace std;

#define FILE\_PATH L"F:/.Уник/3 курс/ОС/9laba/lab09/lab09/OS09\_01.txt"

BOOL insRowFileTxt(LPWSTR FileName, LPWSTR str, DWORD row);

BOOL printFileTxt(LPWSTR FileName);

int main()

{

setlocale(LC\_ALL, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

LPWSTR fileName = (LPWSTR)FILE\_PATH;

printFileTxt(fileName);

insRowFileTxt(fileName, (LPWSTR)L"STR(0)", 0);

insRowFileTxt(fileName, (LPWSTR)L"STR(-1)", -1);

insRowFileTxt(fileName, (LPWSTR)L"STR(15)", 15);

insRowFileTxt(fileName, (LPWSTR)L"STR(7)", 7);

printFileTxt(fileName);

}

BOOL insRowFileTxt(LPWSTR FileName, LPWSTR str, DWORD row)

{

PLARGE\_INTEGER fileSize = new LARGE\_INTEGER();

int rowCount = 1;

int position = 0;

int positionAfter = 0;

bool rowFound = false;

try

{

HANDLE of = CreateFile(

FileName,

GENERIC\_READ | GENERIC\_WRITE,

FILE\_SHARE\_READ,

NULL,

OPEN\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

if (of == INVALID\_HANDLE\_VALUE)

throw "Open file failed";

LockFile(of, 0, 0, 0, 0);

if (GetFileSizeEx(of, fileSize))

{

char\* cstr = new char[wcslen(str) + 1];

wcstombs(cstr, str, strlen(cstr));

char\* buf = new char[fileSize->QuadPart + 1];

char\* bufAfterIns = new char[fileSize->QuadPart + 3 + strlen(cstr)];

ZeroMemory(buf, fileSize->QuadPart + 1);

ZeroMemory(bufAfterIns, fileSize->QuadPart + 3 + strlen(cstr));

DWORD n = NULL;

if (ReadFile(of, buf, fileSize->QuadPart, &n, NULL))

{

buf[fileSize->QuadPart] = '\0';

while (buf[position] != '\0')

{

if ((rowCount == row || row == 0) && !rowFound)

{

for (int i = 0; i < strlen(cstr); i++)

{

bufAfterIns[positionAfter++] = cstr[i];

}

bufAfterIns[positionAfter++] = '\r';

bufAfterIns[positionAfter++] = '\n';

rowFound = true;

rowCount++;

}

if (buf[position] == '\n')

{

rowCount++;

}

bufAfterIns[positionAfter++] = buf[position++];

if (buf[position] == '\0')

{

break;

}

}

if (row == 0 && buf[0] == '\0') {

for (int i = 0; i < strlen(cstr); i++)

{

bufAfterIns[positionAfter++] = cstr[i];

}

rowFound = true;

rowCount++;

}

if (row == MAXDWORD)

{

bufAfterIns[positionAfter++] = '\r';

bufAfterIns[positionAfter++] = '\n';

for (int i = 0; i < strlen(cstr); i++)

{

bufAfterIns[positionAfter++] = cstr[i];

}

rowFound = true;

rowCount++;

}

bufAfterIns[positionAfter] = '\0';

}

if (rowFound)

{

int wideStrLen = MultiByteToWideChar(CP\_UTF8, 0, bufAfterIns, -1, NULL, 0);

wchar\_t\* wideStr = new wchar\_t[wideStrLen];

int result = MultiByteToWideChar(CP\_UTF8, 0, bufAfterIns, -1, wideStr, wideStrLen);

int ansiLen = WideCharToMultiByte(CP\_ACP, 0, wideStr, -1, NULL, 0, NULL, NULL);

char\* ansiStr = new char[ansiLen];

result = WideCharToMultiByte(CP\_ACP, 0, wideStr, -1, ansiStr, ansiLen, NULL, NULL);

printf("\n--- After insertion row #%d:\n%s\n", row == MAXDWORD ? -1 : row, ansiStr);

if (SetFilePointer(of, 0, 0, FILE\_BEGIN) == 0)

{

if(WriteFile(of, bufAfterIns, strlen(bufAfterIns), &n, NULL))

{

printf("\nRow inserted successfully\n");

}

else

{

CloseHandle(of);

throw "WriteFile failed";

}

if (!SetEndOfFile(of))

{

CloseHandle(of);

throw "SetEndOfFile failed";

}

}

else

{

CloseHandle(of);

throw "SetFilePointer failed";

}

}

else

{

CloseHandle(of);

throw "Row isn't found";

}

}

else

{

CloseHandle(of);

throw "GetFileSizeEx failed";

}

CloseHandle(of);

}

catch (const char\* err)

{

std::cout << "--- Error:\n" << err << "\n";

return false;

}

return true;

}

BOOL printFileTxt(LPWSTR FileName)

{

LPBY\_HANDLE\_FILE\_INFORMATION lpFileInformation = new BY\_HANDLE\_FILE\_INFORMATION();

DWORD fileType;

PLARGE\_INTEGER fileSize = new LARGE\_INTEGER();

try

{

HANDLE of = CreateFile(

FileName,

GENERIC\_READ,

NULL,

NULL,

OPEN\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

if (of == INVALID\_HANDLE\_VALUE)

{

CloseHandle(of);

throw "Open file failed";

}

if (GetFileSizeEx(of, fileSize))

{

char\* buf = new char[fileSize->QuadPart \* sizeof(char)];

ZeroMemory(buf, sizeof(buf));

DWORD n = NULL;

if (ReadFile(of, buf, fileSize->QuadPart, &n, NULL))

{

buf[fileSize->QuadPart] = '\0';

int wideStrLen = MultiByteToWideChar(CP\_UTF8, 0, buf, -1, NULL, 0);

wchar\_t\* wideStr = new wchar\_t[wideStrLen];

int result = MultiByteToWideChar(CP\_UTF8, 0, buf, -1, wideStr, wideStrLen);

int ansiLen = WideCharToMultiByte(CP\_ACP, 0, wideStr, -1, NULL, 0, NULL, NULL);

char\* ansiStr = new char[ansiLen];

result = WideCharToMultiByte(CP\_ACP, 0, wideStr, -1, ansiStr, ansiLen, NULL, NULL);

printf("\n-- Read file:%d byte successful:\n%s\n", n, ansiStr);

}

else

{

CloseHandle(of);

throw "ReadFile failed";

}

}

else

{

CloseHandle(of);

throw "GetFileSizeEx failed";

}

CloseHandle(of);

}

catch (const char\* err)

{

std::cout << "--- Error:\n" << err << "\n";

return false;

}

return true;

}

// Блокируем файл для исключения одновременного доступа

// LockFile(of, 0, 0, 0, 0);

// Конвертируем строку в ASCII для корректной работы с буфером

//wcstombs(cstr, str, strlen(cstr));

// Копируем новую строку в буфер после вставки

//for (int i = 0; i < strlen(cstr); i++)

//{

// bufAfterIns[positionAfter++] = cstr[i];

//}

**4---------------------------------------------------------------------------------------------**

#define \_CRT\_NON\_CONFORMING\_WCSTOK

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cstdlib>

#include "Windows.h"

#define FILE\_PATH L"F:/.Уник/3 курс/ОС/9laba/lab09/lab09/OS09\_01.txt"

#define DIR\_PATH L"F:/.Уник/3 курс/ОС/9laba/lab09/lab09"

BOOL printWatchRowFileTxt(LPWSTR FileName, DWORD mlsec);

int rowC = 0;

int main()

{

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

LPWSTR fileName = (LPWSTR)FILE\_PATH;

printWatchRowFileTxt(fileName, 3000);

}

BOOL printWatchRowFileTxt(LPWSTR FileName, DWORD mlsec)

{

PLARGE\_INTEGER fileSize = new LARGE\_INTEGER();

LPWSTR path = (LPWSTR)DIR\_PATH;

char\* cFileName = new char[wcslen(FileName) \* sizeof(char) + 1];

wcstombs(cFileName, FileName, strlen(cFileName));

int i = 0;

int lenName = 0;

printf("File catalog:%s \n", path);

try

{

HANDLE notif = FindFirstChangeNotification(path, false, FILE\_NOTIFY\_CHANGE\_LAST\_WRITE);

DWORD err = GetLastError();

clock\_t t1 = clock();

clock\_t t2 = clock();

DWORD dwWaitStatus;

printf("\nWatch started:", t1);

while (true)

{

dwWaitStatus = WaitForSingleObject(notif, mlsec);

switch (dwWaitStatus)

{

case WAIT\_OBJECT\_0:

{

if (FindNextChangeNotification(notif) == FALSE)

{

break;

}

else

{

Sleep(1000);

int position = 0;

int rowCount = 0;

HANDLE of = CreateFile(

FileName,

GENERIC\_READ,

FILE\_SHARE\_READ,

NULL,

OPEN\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

if (of == INVALID\_HANDLE\_VALUE)

{

CloseHandle(of);

throw "Open file failed";

}

if (GetFileSizeEx(of, fileSize))

{

char\* buf = new char[(fileSize->QuadPart + 1) \* sizeof(char)];

ZeroMemory(buf, (fileSize->QuadPart + 1) \* sizeof(char));

DWORD n = NULL;

if (ReadFile(of, buf, fileSize->QuadPart, &n, NULL))

{

while (buf[position++] != '\0')

{

if (buf[position] == '\n')

{

rowCount++;

}

}

if (position != 0) {

rowCount++;

}

}

}

if (rowC != rowCount)

{

printf("\n Row quantity: %d", rowCount);

rowC = rowCount;

}

CloseHandle(of);

}

}

}

t2 = clock();

if (t2 - t1 > mlsec)

{

break;

}

}

CloseHandle(notif);

printf("\n Watch ended timestamp %d:", t2);

}

catch (const char\* err)

{

std::cout << "--- Error:\n" << err << "\n";

return false;

}

return true;

}