Міністерство освіти і науки, молоді та спорту України

Національний аерокосмічний університет ім. М.Є. Жуковського

Кафедра 503

Лабораторна робота № 1

з дисципліни

«Системне програмування»

Тема: «Изучение Win32 API для получения информации об ошибке, системе, для перекодирования текстов из одного формата в другой»

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Виконав: | ст. гр. 535Б |
|  |  |  | Зленко В.А. |
|  |  | Перевірив: | асистент каф. 503 |
|  |  |  | Мозговий М.В. |

#### Харків 2020

1. **Задание 1**.

Необходимо написать программу, которая бы генерировала ошибку в ходе выполнения системного вызова и выдавала системное описание данной ошибки. Вторым режимом работы данной программы должен быть вывод информации о состоянии системы. Режим запуска программы определяется ключом, передаваемым в командной строке (-e – печать ошибки, -s – печать информации о системе).

1. **Текст программы**

// declaration part

#include <stdio.h>

#include "windows.h"

#include "iostream"

#include <locale.h>

using namespace std;

#define MAX\_ARGUMENT\_LENGTH 2

// functions for implementation -s argument

void ImplementationShowErrorArg();

void ImplementationShowSystemInfoArg();

void StartWithoutArgs();

void GlobalMemoryInfoPrint();

void GetSystemInfoPrint();

// functions for implementation -e argument

bool ThrowOutOfMemoryError();

void ShowError();

// implementation part

int main(int argc, char \*argv[])

{

setlocale(LC\_ALL, "rus");

if (argc != 2)

{

StartWithoutArgs();

}

else

{

if (strcmp(argv[1], "-e") == 0)

{

ImplementationShowErrorArg();

}

else if (strcmp(argv[1], "-s") == 0)

{

ImplementationShowSystemInfoArg();

}

else // error argument

{

StartWithoutArgs();

}

}

return 0;

}

void ImplementationShowErrorArg()

{

// call function for print error

if (ThrowOutOfMemoryError())

{

ShowError();

}

else

{

printf("Sorry, you have too much free memory");

}

}

void ImplementationShowSystemInfoArg()

{

// call function for print information about system

GlobalMemoryInfoPrint();

GetSystemInfoPrint();

}

bool ThrowOutOfMemoryError()

{

DWORD userMemCount = 0;

MEMORYSTATUS memStatus;

GlobalMemoryStatus(&memStatus);

HLOCAL pAllocMem = LocalAlloc(LMEM\_FIXED, memStatus.dwTotalVirtual + 5000000000); // total virtual memory + 5 GB

if (pAllocMem == NULL)

{

return true;

}

else

{

LocalFree(pAllocMem);

return false;

}

}

void ShowError()

{

LPVOID lpMsgBuf;

DWORD dwLastError = GetLastError();

FormatMessage(

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

NULL, dwLastError,

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT),

(LPSTR)& lpMsgBuf, 0, NULL);

printf("\n%s\n", lpMsgBuf);

LocalFree(lpMsgBuf);

return;

}

void StartWithoutArgs()

{

printf("You don`t input argument or input wrong argument. Programm supports 2 arguments:\n");

printf("'-e' - print error\n");

printf("'-s' - print information about system\n\n");

exit(EXIT\_FAILURE);

}

// Func for GlobalMemoryStatus() information print

void GlobalMemoryInfoPrint()

{

MEMORYSTATUS memStatusStruct;

GlobalMemoryStatus(&memStatusStruct);

cout << " \*\*\* information about the current state of both physical and virtual memory \*\*\*" << endl;

cout << "Approximate percentage of physical memory that is in use:"

<< memStatusStruct.dwMemoryLoad << endl;

// phisical memory

cout << "The amount of actual physical memory, in bytes:"

<< memStatusStruct.dwTotalPhys << endl;

cout << "The amount of physical memory currently available, in bytes:"

<< memStatusStruct.dwAvailPhys << endl;

// memory limit for process

cout << "The current size of the committed memory limit, in bytes:"

<< memStatusStruct.dwTotalPageFile << endl;

cout << "The maximum amount of memory the current process can commit, in bytes:"

<< memStatusStruct.dwAvailPageFile << endl;

// virtual memory

cout << "The total amount of virtual memory, in bytes:"

<< memStatusStruct.dwTotalVirtual << endl;

cout << "The amount of availvable virtual memory, in bytes:"

<< memStatusStruct.dwAvailVirtual << endl;

cout << "\nUsed function is GlobalMemoryStatus()\n" << endl;

}

// Func for GetSystemInfo information print

void GetSystemInfoPrint()

{

SYSTEM\_INFO sysInfo;

GetSystemInfo(&sysInfo);

printf(" \*\*\* Hardware information \*\*\*\n");

printf(" Processor architecture: %u\n", sysInfo.wProcessorArchitecture);

printf(" Processor type: %u\n", sysInfo.dwProcessorType);

printf(" Number of processors: %u\n", sysInfo.dwNumberOfProcessors);

printf(" Active processor mask: %u\n", sysInfo.dwActiveProcessorMask);

printf(" Page size: %u\n", sysInfo.dwPageSize);

printf(" Minimum application address: %p\n", sysInfo.lpMinimumApplicationAddress);

printf(" Maximum application address: %p\n", sysInfo.lpMaximumApplicationAddress);

printf(" The granularity for the starting"

" address at which virtual"

" memory can be allocated: %u\n", sysInfo.dwAllocationGranularity);

printf("\nUsed function is GetSystemInfo()\n");

}

1. **Тестирование**

Таблица 1 – тестовые случаи

|  |  |  |
| --- | --- | --- |
| Название теста | Входные данные | Выходные данные |
| Проверка работоспособности аргумента ‘-e’ | Аргумент ‘-e’ при запуске программы из командной строки | Сообщение об ошибке, представлено на  Рисунке 1 |
| Проверка работоспособности аргумента ‘-s’ | Аргумент ‘-s’ при запуске программы из командной строки | Описание системы, представлено на  Рисунке 2 |

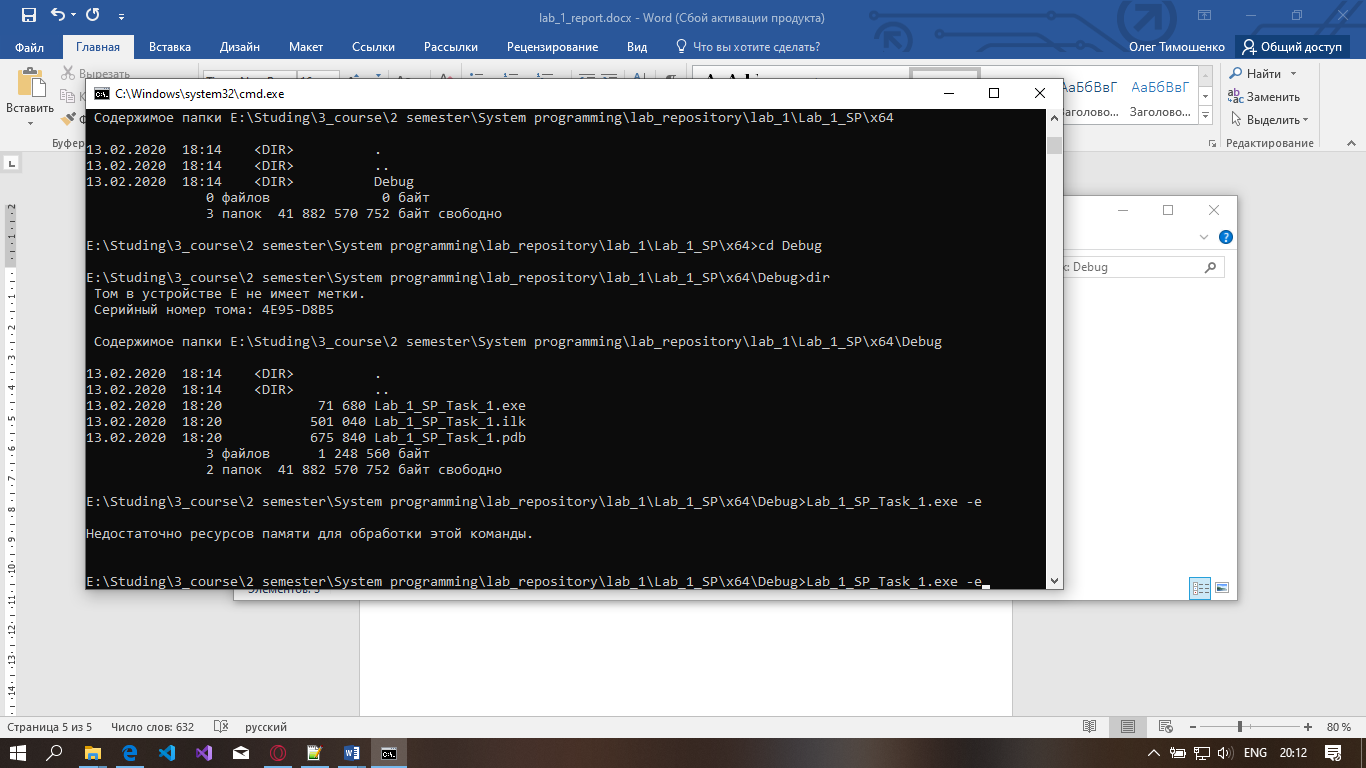


Рисунок 1 – результат работы программы для задания 1 с аргументом ‘-е’

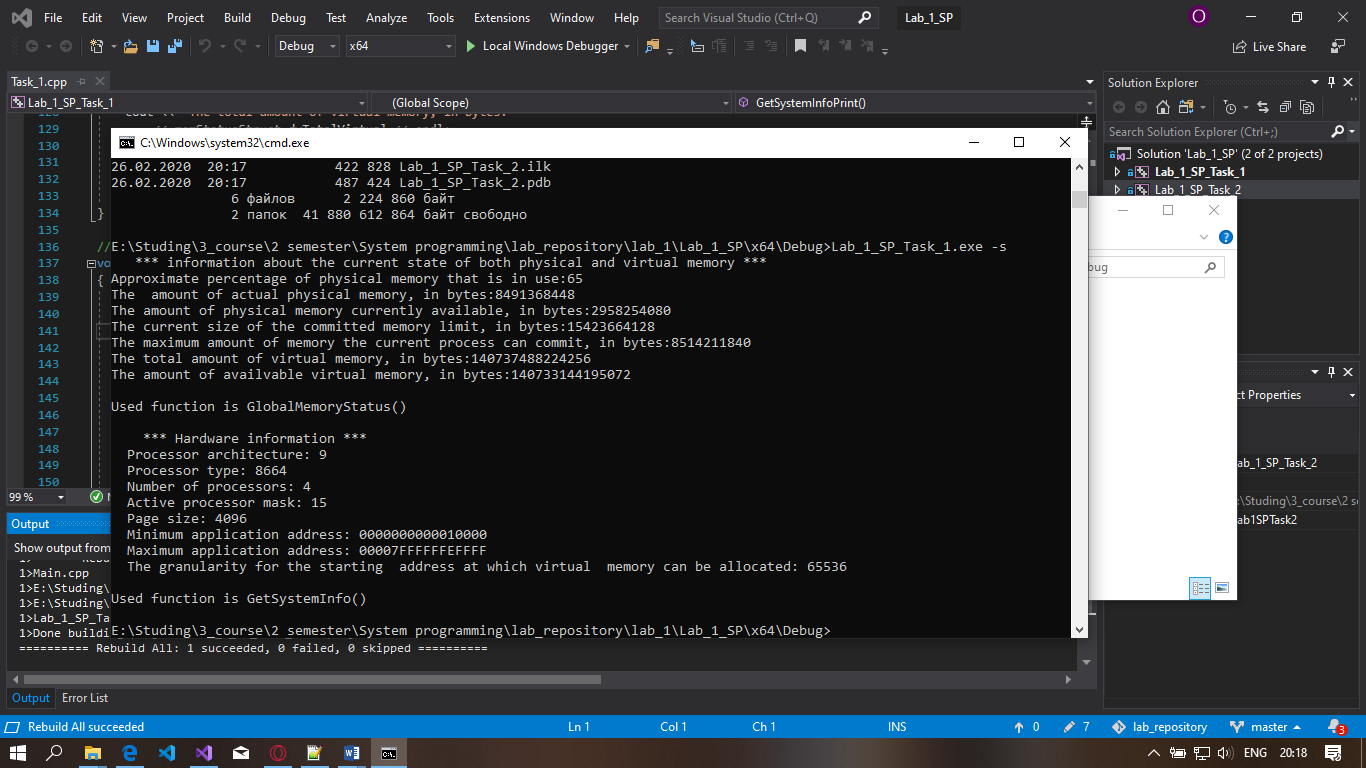


Рисунок 1 – результат работы программы для задания 1 с аргументом ‘-s’

1. **Задание 2**.

Написать программу, позволяющую выполнять перекодировку текста из ASCII в Юникод и обратно. В качестве входных данных выступает файл с текстом. Программа при запуске получает параметр командной строки определяющий исходную кодировку файла (-a – ANSI файл, -u – Юникод файл).

1. **Текст программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include "windows.h"

#include <locale.h>

#include <iostream>

using namespace std;

#define ANSI\_KEY "-a"

#define UNICODE\_KEY "-u"

#define SIZE\_BUFF\_FOR\_READING\_FILE 2048

void ANSI\_To\_Unicode(LPCSTR);

void Unicode\_To\_ANSI(LPCSTR);

void ShowError();

bool ConcatFileNameWithString(char\* FileName, char\* StringToConcat, char\* resultString, int len\_resultString);

int main(int argc, char\* argv[])

{

setlocale(LC\_CTYPE, "rus");

if (argc != 3)

{

fprintf(stderr, "Ошибка! Не переданы все аргументы дял программы\n");

fprintf(stderr, "Необходимая форма: ""имя\_программы ключ\_для\_кодировки имя\_файла\_для\_перекодирования\n");

fprintf(stderr, "Ключи для кодировки: '-a' - текст в ANSI кодировке\n");

fprintf(stderr, "Ключи для кодировки: '-u' - текст в Unicode кодировке\n");

system("Pause");

return 0;

}

else

{

if (strcmp(argv[1], ANSI\_KEY) == 0)

{

ANSI\_To\_Unicode(argv[2]);

}

else if (strcmp(argv[1], UNICODE\_KEY) == 0)

{

Unicode\_To\_ANSI(argv[2]);

}

else

{

fprintf(stderr, "Ошибка ввода параметров\n");

}

}

fprintf(stdout, " \*\*\*Конвертация была успешно выполнена \*\*\* ");

return 0;

}

void ANSI\_To\_Unicode(LPCSTR fileName)

{

HANDLE handleSourceFile, handleDestinationFile; // handles to files

CHAR rdBuff[SIZE\_BUFF\_FOR\_READING\_FILE]; // buff to read text from source file

WCHAR buffCharsEncodedToWChar\_t[SIZE\_BUFF\_FOR\_READING\_FILE]; // buff contain encoded ASCI text to Unicode

DWORD countReadedChars, countWritenChars; // counters of read Symbols(in ReadFIle) and writen Symbols(in WriteFile)

BOOL resultRdFile = true;

char additionalTextToFileName[] = "\_encoded\_To\_Unicode";

char nameDestinationFile[FILENAME\_MAX]; // name to file with recoded text

ConcatFileNameWithString((char\*)fileName, additionalTextToFileName, nameDestinationFile, FILENAME\_MAX);

printf(" \*\*\* Перекодированый текст будет находится в \*\*\* \n %s\n", nameDestinationFile);

// OPEN FILES

handleSourceFile = CreateFile(fileName, GENERIC\_READ, FILE\_SHARE\_READ,

NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (handleSourceFile == INVALID\_HANDLE\_VALUE)

{

ShowError();

exit(EXIT\_FAILURE);

}

handleDestinationFile = CreateFile(nameDestinationFile, GENERIC\_WRITE, 0,

NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (handleDestinationFile == INVALID\_HANDLE\_VALUE)

{

ShowError();

exit(EXIT\_FAILURE);

}

// In cycle read all file in blocks of size - SIZE\_BUFF\_FOR\_READING\_FILE

while (((resultRdFile = ReadFile(handleSourceFile, &rdBuff, sizeof(rdBuff), &countReadedChars, NULL)) != 0) && (countReadedChars != 0))

{

// recode text to Unicode from ASCI

MultiByteToWideChar(GetACP(), 0, rdBuff, countReadedChars, buffCharsEncodedToWChar\_t, SIZE\_BUFF\_FOR\_READING\_FILE);

// write recoded block to new file

if (!WriteFile(handleDestinationFile, buffCharsEncodedToWChar\_t, countReadedChars\*2, &countWritenChars, NULL))

{

ShowError();

exit(EXIT\_FAILURE);

}

}

if (resultRdFile == false && countReadedChars != 0)

{

ShowError();

exit(EXIT\_FAILURE);

}

CloseHandle(handleSourceFile);

CloseHandle(handleDestinationFile);

}

void Unicode\_To\_ANSI(LPCSTR fileName)

{

HANDLE handleSourceFile, handleDestinationFile; // handles to files

WCHAR rdBuff[SIZE\_BUFF\_FOR\_READING\_FILE]; // buff to read text from source file

CHAR buffWCHARSEncodedToCHAR[SIZE\_BUFF\_FOR\_READING\_FILE]; // buff contain encoded ASCI text to Unicode

CCHAR unknownSymb(254);

BOOL isUsedUnknownSymb = false;

DWORD countReadedBytes, countWritenBytes; // counters of read Symbols(in ReadFIle) and writen Symbols(in WriteFile)

BOOL resultRdFile = true;

char additionalTextToFileName[] = "\_encoded\_To\_ANSI";

char nameDestinationFile[FILENAME\_MAX]; // name to file with recoded text

ConcatFileNameWithString((char\*)fileName, additionalTextToFileName, nameDestinationFile, FILENAME\_MAX);

printf(" \*\*\* Перекодированый текст будет находится в \*\*\* \n %s\n", nameDestinationFile);

// OPEN FILES

handleSourceFile = CreateFile(fileName, GENERIC\_READ, FILE\_SHARE\_READ,

NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (handleSourceFile == INVALID\_HANDLE\_VALUE)

{

ShowError();

exit(EXIT\_FAILURE);

}

handleDestinationFile = CreateFile(nameDestinationFile, GENERIC\_WRITE, 0,

NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (handleDestinationFile == INVALID\_HANDLE\_VALUE)

{

ShowError();

exit(EXIT\_FAILURE);

}

if (!ReadFile(handleSourceFile, &rdBuff, 2, &countReadedBytes, NULL))

{

ShowError();

exit(EXIT\_FAILURE);

}

// In cycle read all file in blocks of size - SIZE\_BUFF\_FOR\_READING\_FILE

while (((resultRdFile = ReadFile(handleSourceFile, &rdBuff, sizeof(rdBuff), &countReadedBytes, NULL)) != 0) && (countReadedBytes != 0))

{

WideCharToMultiByte(GetACP(), 0, rdBuff, countReadedBytes / sizeof(rdBuff[0]), buffWCHARSEncodedToCHAR,

countReadedBytes / sizeof(rdBuff[0]), NULL, NULL);

// write recoded block to new file

if (!WriteFile(handleDestinationFile, buffWCHARSEncodedToCHAR, countReadedBytes / sizeof(rdBuff[0]), &countWritenBytes, NULL))

{

ShowError();

exit(EXIT\_FAILURE);

}

}

if (resultRdFile == false && countReadedBytes != 0)

{

ShowError();

exit(EXIT\_FAILURE);

}

CloseHandle(handleSourceFile);

CloseHandle(handleDestinationFile);

}

// This function concat file name (file name with format: .txt, .xml etc) with string,

// this string insert in end of file name, but before format (i.e. before last dot in file name string)

bool ConcatFileNameWithString(char \* FileName, char \* StringToConcat, char \* resultString, int len\_resultString)

{

for (int i = 0; i < len\_resultString; i++)

{

resultString[i] = '\0';

}

if ((strlen(FileName) + strlen(StringToConcat)) > FILENAME\_MAX)

{

resultString = NULL;

return false;

}

char \* refToLastDot = strrchr(FileName, '.');

int lastDotPositionInFileName = refToLastDot - FileName;

strncpy(resultString, FileName, lastDotPositionInFileName);

snprintf(resultString, FILENAME\_MAX, "%s%s%s", resultString, StringToConcat, refToLastDot);

return true;

}

void ShowError()

{

LPVOID lpMsgBuf;

DWORD dwLastError = GetLastError();

FormatMessage(

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

NULL, dwLastError,

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT),

(LPSTR)& lpMsgBuf, 0, NULL);

printf("\n%s\n", lpMsgBuf);

LocalFree(lpMsgBuf);

return;

}

1. **Тестирование**

Таблица 2 – тестовые случаи

|  |  |  |
| --- | --- | --- |
| Название теста | Входные данные | Выходные данные |
| Проверка работоспособности аргумента ‘-a’ | Аргумент ‘-a’ и путь к ANSI файлу при запуске программы из командной строки | Перекодирование ANSI файла в новый Unicode файл, представлено на Рисунках 3, 4 |
| Проверка работоспособности аргумента ‘-u’ | Аргумент ‘-u’ и путь к Unicode файлу при запуске программы из командной строки | Перекодирование Unicode файла в новый ANSI файл, представлено на Рисунках 5, 6 |

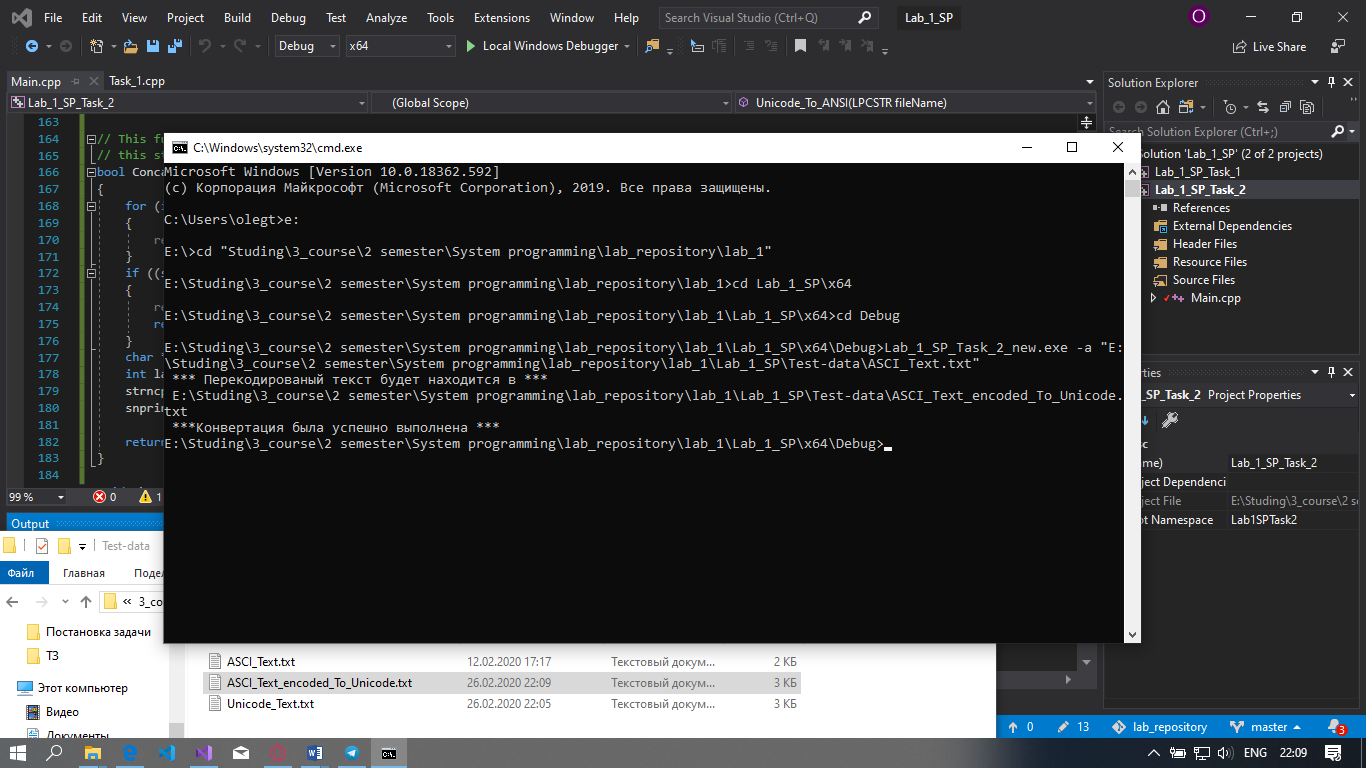


Рисунок 3 – результат работы программы для задания 1 с аргументом ‘-е’

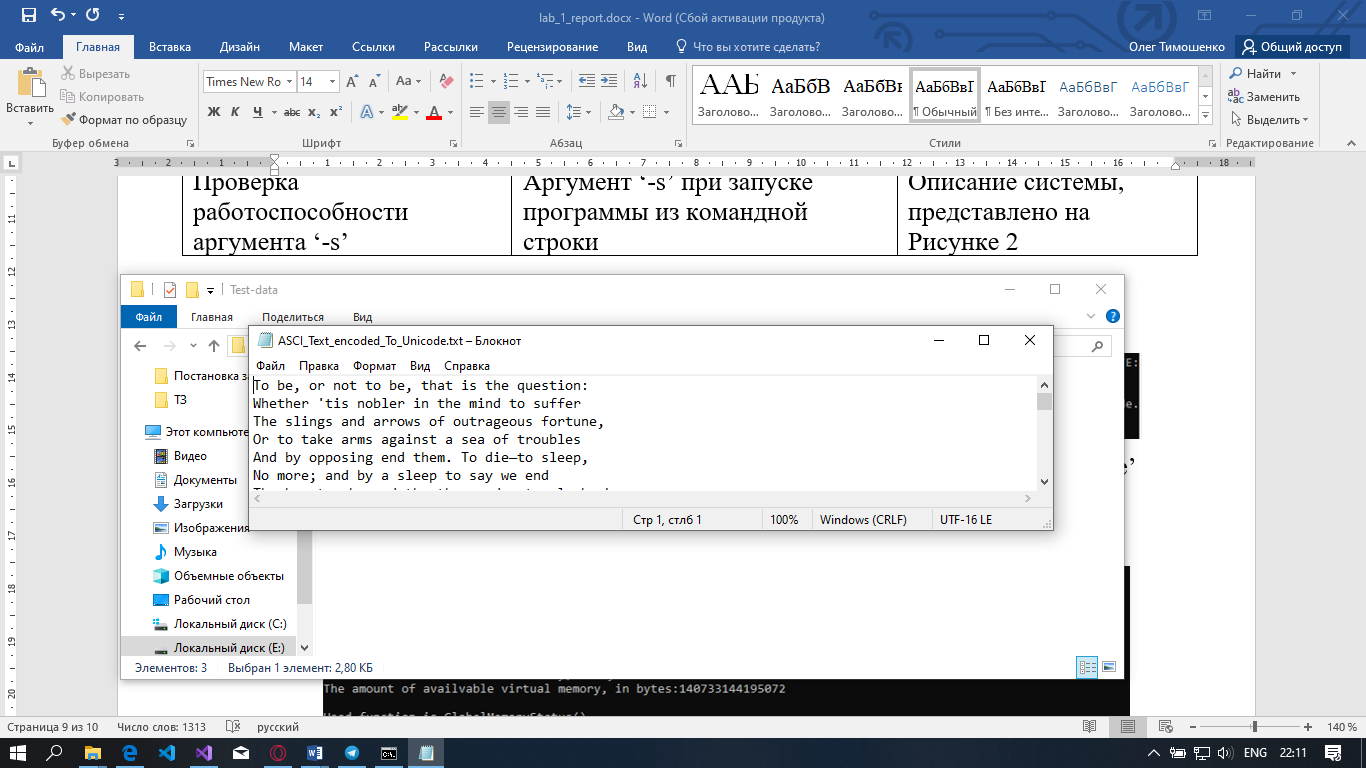


Рисунок 4 – часть файла полученная в результате работы программы задания 1 с аргументом ‘-е’

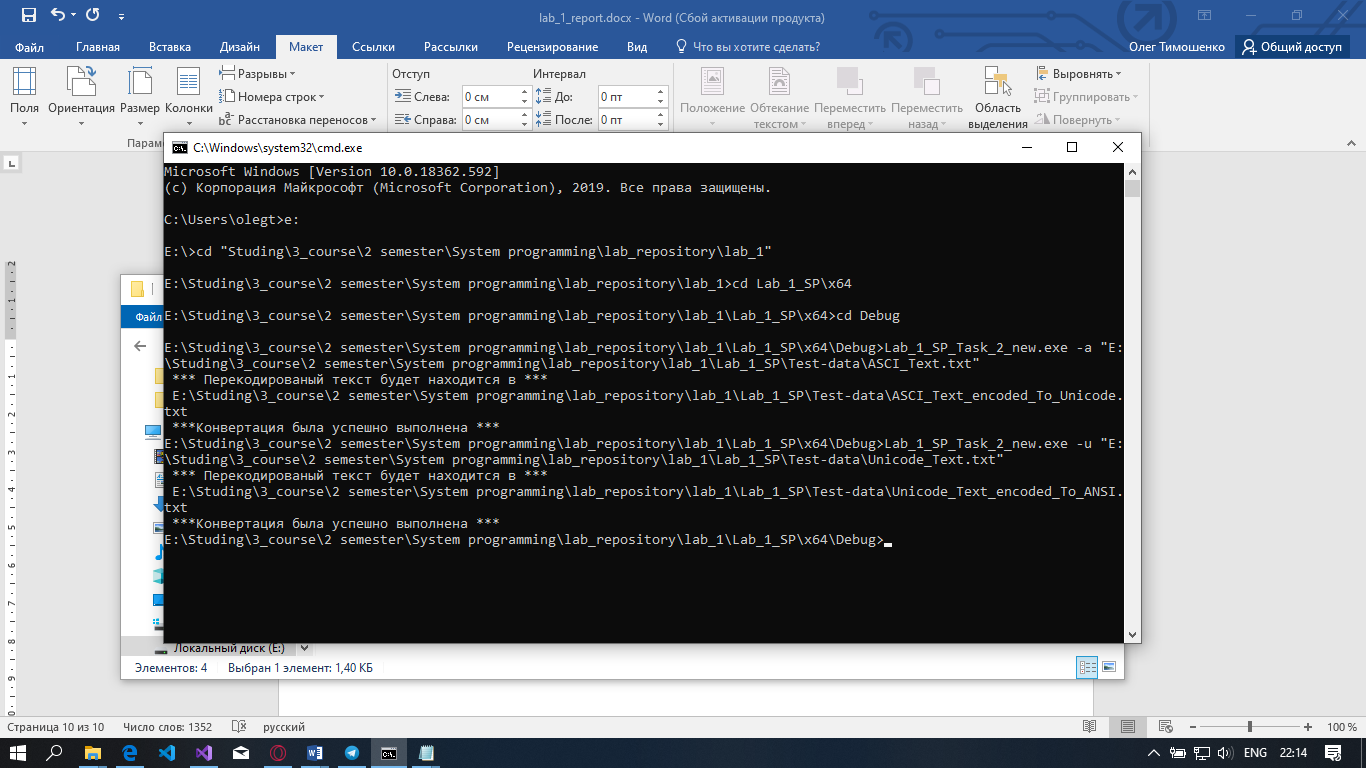


Рисунок 5 – результат работы программы для задания 1 с аргументом ‘-u’

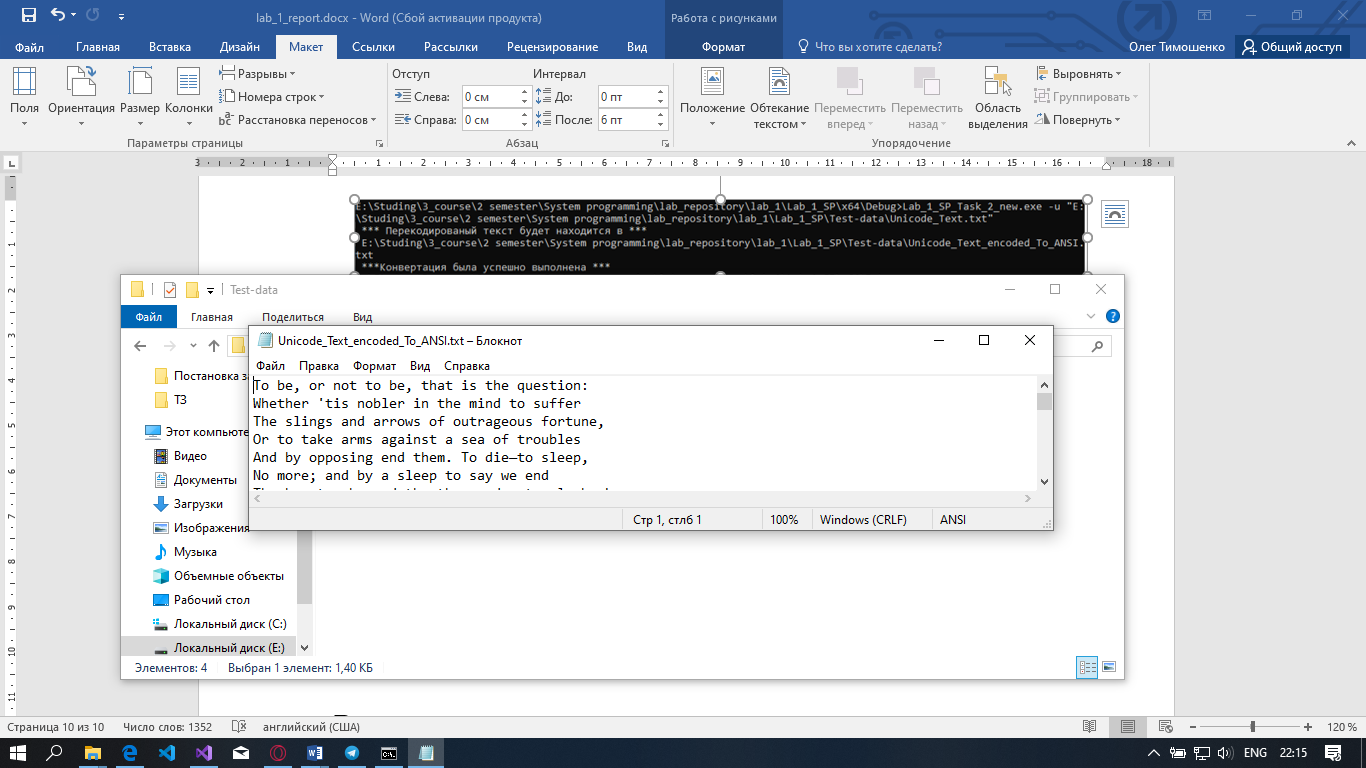


Рисунок 6 – часть файла полученная в результате работы программы задания 1 с аргументом ‘-u’

**Выводы**

В ходе выполнения данной лабораторной работы я изучил вызовы Win32 API, которые позволяют нам получить информацию об ошибке, а также были изучены функции которые позволяют нам собрать информацию о системе. И они помогают нам перекодировать текст из ASPI в Unicode и вернуть всё обратно.