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 |
|  |  |  | Жовнір В.Е |
|  |  | 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 |

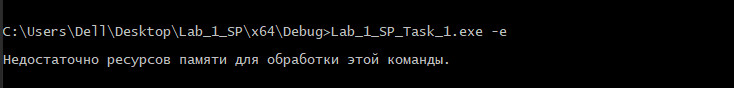
****

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

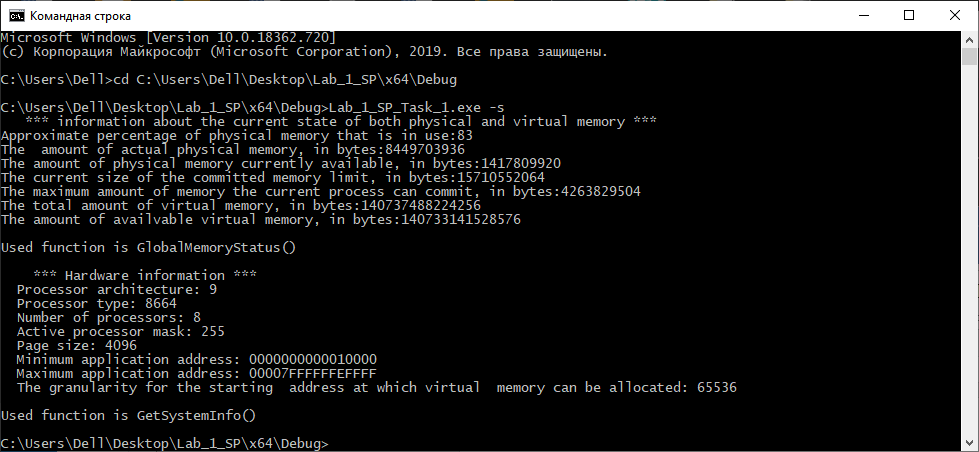


Рисунок 1 – результат работы программы для задания 1 с аргументом ‘-s’

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 1024

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\_ALL, "rus");

    if (argc != 3)

    {

        fprintf(stderr, "Error! Incorrect arguments count for programm\n");

        fprintf(stderr, "Necessary keys to run programm: ""-codepage\_key filename\_for\_recoding\n");

        fprintf(stderr, "Codepage key: '-a' - ANSI input\n");

        fprintf(stderr, "Codepage key: '-u' - Unicode input\n");

        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, "Incorrect arguments\n");

        }

    }

    fprintf(stdout, " \*\*\* Recoding completed successfully \*\*\* ");

    return 0;

}

void ANSI\_To\_Unicode(LPCSTR fileName)

{

    HANDLE handleSourceFile, handleDestinationFile; // handles to files

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

    WCHAR \* unicodeBuff; // buff contain encoded ASCI text to Unicode

    DWORD countRdChars, countWrWChars; // 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(" \*\*\* Recoded file will be located in \*\*\* \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, &asciBuff, sizeof(asciBuff), &countRdChars, NULL)) != false) && (countRdChars != 0))

    {

        // find necessary buff size for recoded text

        int sizeToAllocate = MultiByteToWideChar(CP\_UTF8, 0, asciBuff, countRdChars, NULL, 0);

        // allocate necessary buff

        unicodeBuff = (WCHAR \*)calloc(sizeToAllocate, sizeof(WCHAR));

        // recode text to Unicode from ASCI

        MultiByteToWideChar(CP\_UTF8, 0, asciBuff, countRdChars, unicodeBuff, sizeToAllocate);

        // write recoded block to new file

        if ((!WriteFile(handleDestinationFile, unicodeBuff, sizeToAllocate \* sizeof(WCHAR), &countWrWChars, NULL)) ||

            (countWrWChars != sizeToAllocate \* sizeof(WCHAR)))

        {

            ShowError();

            exit(EXIT\_FAILURE);

        }

        free(unicodeBuff);

    }

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

    {

        ShowError();

        exit(EXIT\_FAILURE);

    }

    CloseHandle(handleSourceFile);

    CloseHandle(handleDestinationFile);

}

void Unicode\_To\_ANSI(LPCSTR fileName)

{

    HANDLE handleSourceFile, handleDestinationFile; // handles to files

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

    WCHAR \* unicodeBuff; // buff to contain encoded raw bytes to wide char

    CHAR \* asciBuff; // buff contain encoded ASCI text to Unicode

    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(" \*\*\* Recoded file will be located in \*\*\* \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);

    }

    // read first 2 bytes which mean ending style (BE or LE), because it doesnt need for ansi files

    if (!ReadFile(handleSourceFile, &rawBytesBuff, 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, &rawBytesBuff, sizeof(rawBytesBuff), &countReadedBytes, NULL)) != false) && (countReadedBytes != 0))

    {

        // Read raw bytes, than get count of unicode symbols in this raw bytes

        int unicodeLenght = MultiByteToWideChar(CP\_UTF8, 0, rawBytesBuff, countReadedBytes, NULL, 0);

        // Allocate buff for containing raw bytes encoded to wide char

        unicodeBuff = (WCHAR\*)calloc(unicodeLenght, sizeof(WCHAR));

        // Encode raw bytes to wide char

        MultiByteToWideChar(CP\_UTF8, 0, rawBytesBuff, countReadedBytes,

                            unicodeBuff, unicodeLenght);

        // Get necessary count bytes to contain convert wide char array to multibyte array

        int asciLenght = WideCharToMultiByte(CP\_ACP, 0, unicodeBuff, unicodeLenght, NULL, 0, NULL, NULL);

        // Allocate necessary buff

        asciBuff = (CHAR \*) calloc(asciLenght, sizeof(CHAR));

        // convert wide char to multibyte

        WideCharToMultiByte(CP\_ACP, 0, unicodeBuff, unicodeLenght, asciBuff,

                            asciLenght, NULL, NULL);

        // write recoded block to new file

        if ((!WriteFile(handleDestinationFile, asciBuff, asciLenght, &countWritenBytes, NULL)) || (countWritenBytes != asciLenght))

        {

            ShowError();

            exit(EXIT\_FAILURE);

        }

        free(unicodeBuff);

        free(asciBuff);

    }

    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 |

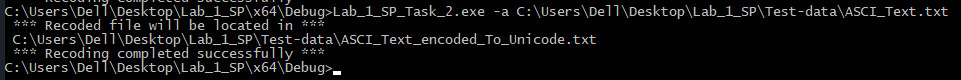
****

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

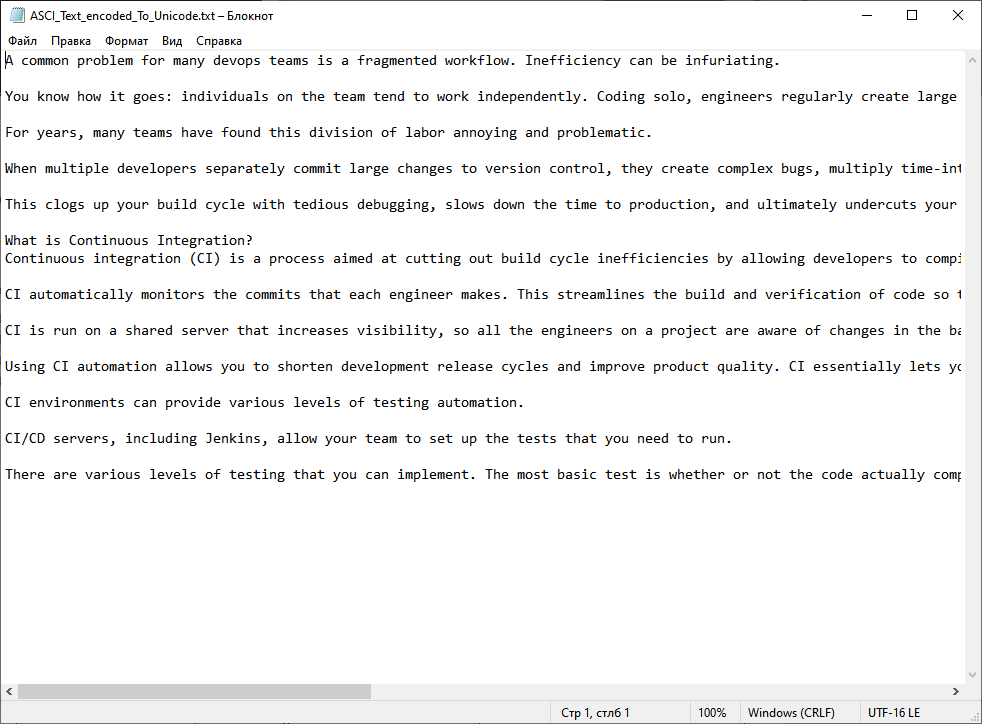


Рисунок 4 – часть файла полученная в результате работы программы задания 1 с аргументом ‘-a’

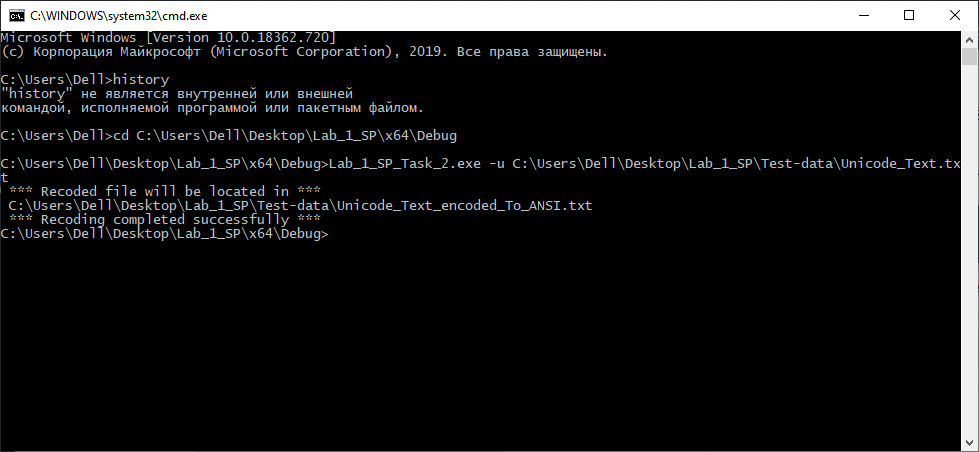


Рисунок 5 – результат работы программы для задания 1 с аргументом ‘-u’

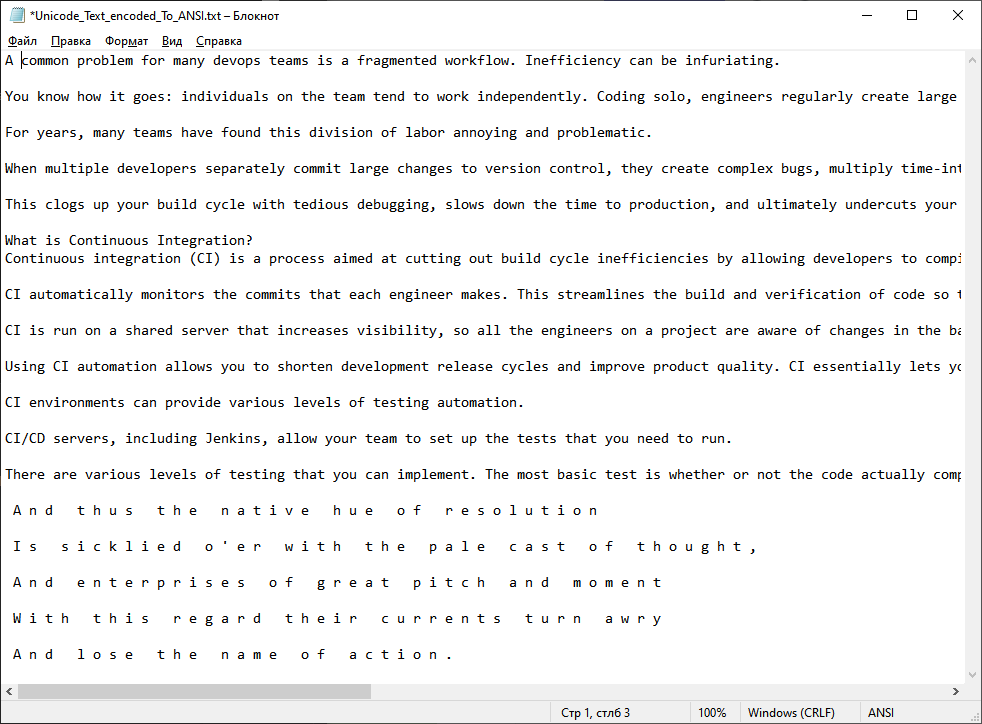


Рисунок 6 – часть файла полученная в результате работы программы задания 1 с аргументом ‘-u’

**Выводы**

В результате выполнения лабораторной работы были изучены системные вызовы Win32 API, позволяющие получить информацию об ошибке, изучены функции сбора информации о системе, перекодировки текста из ASCI в Unicode и наоборот.