МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп'ютерних систем та інфокомунікацій

Кафедра комп'ютерних систем, мереж і кібербезпеки

**Лабораторна робота№1**

з дисципліни «Системное программирование»

(назва дисципліни)

на тему: «Изучение системных вызовов Win32 API, позволяющих получить информацию об ошибке.»

Виконав: студент 3 курсу групи № 535а

напряму підготовки (спеціальності)

503 Компьютерні системі та мережі\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр і назва напряму підготовки (спеціальності))

Маловічко Є.В.

(прізвище й ініціали студента)

Прийняв: асистент каф.503

Мозговий М.В.

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала: \_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_

Харків – 2020

**Цель работы:**

1. Изучение системных вызовов Win32 API, позволяющих получить информацию об ошибке.
2. Изучение функций сбора информации о системе.

**Постановка задачи:**

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).
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 Код программы:**

#include <iostream>

#include <stdio.h>

#include <Windows.h>

#include <locale.h>

using namespace std;

void getInfo() {

SYSTEM\_INFO systemInfo;

GetSystemInfo(&systemInfo);

cout << "Tool information:" << endl;

cout << "Tool information:" << systemInfo.dwOemId << endl;

cout << "Number of processors:" << systemInfo.dwNumberOfProcessors << endl;

cout << "Page size:" << systemInfo.dwPageSize << endl;

cout << "Processor type:" << systemInfo.dwProcessorType << endl;

cout << "Min Application Number:" << systemInfo.lpMinApplicationNumber << endl;

cout << "Max application Number:" << systemInfo.lpMaxApplicationNumber << endl;

cout << "Active processor mask:" << systemInfo.dwActiveProcessorMask << endl;

exit(0);

}

void generateError() {

setlocale(LC\_ALL, "Russian");

MEMORYSTATUS memoryInfo;

GlobalMemoryStatus(&memoryInfo);

HLOCAL allocated\_memory\_object = LocalAlloc(LPTR, memoryInfo.dwTotalVirtual \* 1000);

if (allocated\_memory\_object == NULL) {

LPVOID errorMessage;

DWORD eCode = GetLastError();

FormatMessage(FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM, NULL,

eCode, MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), (LPTSTR)&errorMessage, 0, NULL);

char\* \_errorMessage = (char\*)errorMessage;

wprintf(L"ERROR was generated: failed with 0x%x code\n", eCode);

wprintf(L"ERROR formatted message: %s\n", \_errorMessage);

}

exit(0);

}

void incorrectInput() {

cout << "You made a mistake" << endl;

cout << "There are only two arguments available:" << endl;

cout << "-e = generate error & print error" << endl;

cout << "-s = print system information" << endl;

exit(0);

}

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

if (argc != 2 || ((strcmp(argv[1], "-e") != 0) && (strcmp(argv[1], "-s"))) != 0) {

incorrectInput();

}

else {

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

generateError();

}

else {

getInfo();

}

}

}

**Результат работы:**

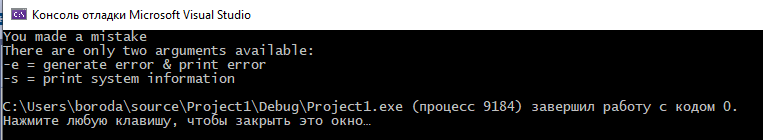


Рисунок 1 – Результат отработки программы в трех случаях, без параметров и с параметрами -s и -e.

**2 Код программы:**

#include <stdio.h>

#include "windows.h"

#include <locale.h>

#include <iostream>

using namespace std;

void PrintError()

{

LPVOID lpvMessage;

DWORD dwError = GetLastError();

FormatMessage(FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM, NULL, dwError, MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), (LPWSTR)&lpvMessage, 0, NULL);

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

LocalFree(lpvMessage);

return;

}

void EncodeToANSIFromUnicode(LPCSTR fileName)

{

HANDLE sourceFile, destinationFile;

CHAR buffer[1024];

WCHAR\* encodedCharsToWChar\_t;

DWORD readedChars, writtenChars;

BOOL readFile = true;

char nameOfDestinationFile[FILENAME\_MAX];

snprintf(nameOfDestinationFile, sizeof(nameOfDestinationFile), "%s%s", "UnicodeEncoded", fileName);

printf("Результат находится в \n %s\n", nameOfDestinationFile);

sourceFile = CreateFileA(fileName, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (sourceFile == INVALID\_HANDLE\_VALUE)

{

PrintError();

printf("1 error\n");

exit(EXIT\_FAILURE);

}

destinationFile = CreateFileA(nameOfDestinationFile, GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (destinationFile == INVALID\_HANDLE\_VALUE)

{

PrintError();

printf("2 error\n");

exit(EXIT\_FAILURE);

}

while (((readFile = ReadFile(sourceFile, &buffer, sizeof(buffer), &readedChars, NULL)) != 0) && (readedChars != 0))

{

int allocationSize = MultiByteToWideChar(GetACP(), 0, buffer, readedChars, NULL, 0);

encodedCharsToWChar\_t = (WCHAR\*)calloc(allocationSize, sizeof(WCHAR));

MultiByteToWideChar(GetACP(), 0, buffer, readedChars, encodedCharsToWChar\_t, allocationSize);

if (!WriteFile(destinationFile, encodedCharsToWChar\_t, readedChars \* 2, &writtenChars, NULL))

{

PrintError();

printf("3 error\n");

exit(EXIT\_FAILURE);

}

free(encodedCharsToWChar\_t);

}

if (readFile == false && readedChars != 0)

{

PrintError();

printf("4 error\n");

exit(EXIT\_FAILURE);

}

CloseHandle(sourceFile);

CloseHandle(destinationFile);

}

void EncodeToUnicodeFromANSI(LPCSTR fileName)

{

HANDLE sourceFile, destinationFile;

CHAR buffer[1024];

WCHAR\* bytesEncodedIntoWchar;

CHAR\* WCharsEncodedToCHARS;

BOOL isSymbolUnknown = false;

DWORD readedBytes, writtenBytes;

BOOL readFile = true;

char nameDestinationFile[FILENAME\_MAX];

snprintf(nameDestinationFile, sizeof(nameDestinationFile), "%s%s", "ASCIEncoded", fileName);

printf("Результат находится в \n %s\n", nameDestinationFile);

sourceFile = CreateFileA(fileName, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (sourceFile == INVALID\_HANDLE\_VALUE)

{

PrintError();

exit(EXIT\_FAILURE);

}

destinationFile = CreateFileA(nameDestinationFile, GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (destinationFile == INVALID\_HANDLE\_VALUE)

{

PrintError();

exit(EXIT\_FAILURE);

}

if (!ReadFile(sourceFile, &buffer, 2, &readedBytes, NULL))

{

PrintError();

exit(EXIT\_FAILURE);

}

while (((readFile = ReadFile(sourceFile, &buffer, sizeof(buffer), &readedBytes, NULL)) != false) && (readedBytes != 0))

{

int symbols = MultiByteToWideChar(GetACP(), 0, buffer, readedBytes, NULL, 0);

bytesEncodedIntoWchar = (WCHAR\*)calloc(symbols, sizeof(WCHAR));

MultiByteToWideChar(GetACP(), 0, buffer, readedBytes, bytesEncodedIntoWchar, symbols);

int allocationSize = WideCharToMultiByte(CP\_ACP, 0, bytesEncodedIntoWchar, symbols, NULL, 0, NULL, NULL);

WCharsEncodedToCHARS = (CHAR\*)calloc(allocationSize, sizeof(CHAR));

WideCharToMultiByte(CP\_ACP, 0, bytesEncodedIntoWchar, symbols, WCharsEncodedToCHARS, allocationSize, NULL, NULL);

if (!WriteFile(destinationFile, WCharsEncodedToCHARS, allocationSize, &writtenBytes, NULL))

{

PrintError();

exit(EXIT\_FAILURE);

}

free(bytesEncodedIntoWchar);

free(WCharsEncodedToCHARS);

}

if (readFile == false && readedBytes != 0)

{

PrintError();

exit(EXIT\_FAILURE);

}

CloseHandle(sourceFile);

CloseHandle(destinationFile);

}

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

{

setlocale(LC\_ALL, "rus");

if (argc != 3)

{

fprintf(stderr, "\*\*\*Error in arguments!\*\*\*\n");

fprintf(stderr, "-a - ANSI\n");

fprintf(stderr, "-u - Unicode\n");

system("Pause");

return 0;

}

else

{

printf("%s", argv[1]);

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

{

EncodeToANSIFromUnicode(argv[2]);

}

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

{

EncodeToUnicodeFromANSI(argv[2]);

}

else

{

fprintf(stderr, "\*\*\*Input Error!!!\*\*\*\n");

}

}

fprintf(stdout, "Success!");

return 0;

}

**Результат работы:**

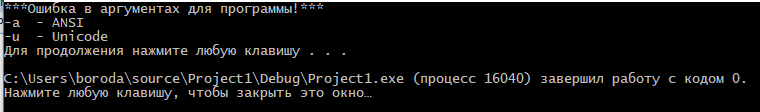


Рисунок 2 – Исходный файл результат

**Выводы**

В данной лабораторной работемы изучили системные вызовы Win32 API, позволяющие получить информацию об ошибке. А так же изучили функции сбора информации о системе.