МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ  
Национальный аэрокосмический университет им. Н.Е. Жуковского  
«Харьковский авиационный институт»

Кафедра компьютерных систем и сетей

Лабораторная работа № 1

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

по дисциплине «Системное программирование»

ХАИ.503.535а.17В.050102, 1405081 ПЗ

Выполнил студент гр. 535а Курта Р. В

(подпись, дата)

Проверил ассистент каф. 503  
(научная степень, ученое звание, должность)

Трубилко А. В.  
(подпись, дата) (Ф.И.О)

Харьков 2017

**Цель работы:**

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

Постановка задачи:

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).

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).

## Выполнение работы:

Программа может иметь оконный или консольный интерфейс. Для решения поставленной задачи в качестве способа сгенерировать ошибку может быть использован системный вызов для выделения памяти LocalAlloc(). Для сбора информации должны быть задействованы функции:

* 1. GlobalMemoryStatus()
  2. GetSystemInfo()

Программа должна иметь консольный интерфейс, где в командной строке будет передаваться имя файла, который нужно перекодировать. Перекодирование текстовой информации должно выполняться с применением системных вызовов:

* 1. MultiByteToWideChar()
  2. WideCharToMultiByte()

**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).

**Текст программы**

В среде Microsoft Visual Studio была написана программа приведенная ниже.

#include "stdafx.h"

void causeError();

void printErrorMessage(HRESULT hResult = NULL);

void printSystemInformation();

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

{

setlocale(LC\_ALL, "Rus");

if (argc != 2)

{

printf("Неправильные аргументы\n");

}

else if (!strcmp(argv[1], "-e"))

{

causeError();

printErrorMessage();

}

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

{

printSystemInformation();

}

else

{

printf("Неправильные аргументы\n");

}

return 0;

}

void causeError()

{

/\*CreateFile(L"", GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);\*/

LocalAlloc(LPTR, ULONG\_MAX);

}

void printErrorMessage(HRESULT hResult)

{

if (hResult == NULL)

{

hResult = GetLastError();

}

LPTSTR errorText = NULL;

FormatMessage(

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

NULL,

hResult,

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT),

(LPTSTR)&errorText,

0,

NULL);

if (NULL != errorText)

{

wprintf(L"%s", errorText);

LocalFree(errorText);

}

}

void printSystemInformation()

{

printf("\*\*\*\*Информация о процессоре\*\*\*\*\n");

SYSTEM\_INFO systeminfo;

GetNativeSystemInfo(&systeminfo);

CHAR szProcArchitecture[19];

switch (systeminfo.wProcessorArchitecture)

{

case PROCESSOR\_ARCHITECTURE\_AMD64:

strcpy(szProcArchitecture, "x64");

break;

case PROCESSOR\_ARCHITECTURE\_ARM:

strcpy(szProcArchitecture, "ARM");

break;

case PROCESSOR\_ARCHITECTURE\_IA64:

strcpy(szProcArchitecture, "Intel Itanium-based");

break;

case PROCESSOR\_ARCHITECTURE\_INTEL:

strcpy(szProcArchitecture, "x86");

break;

default:

strcpy(szProcArchitecture, "Unknown");

break;

}

printf("Архитектура процессора: %s\n", szProcArchitecture);

printf("Уровень процессора: %u\n", systeminfo.wProcessorLevel);

printf("Размер страницы: %u\n", systeminfo.dwPageSize);

printf("Количество процессоров: %u\n", systeminfo.dwNumberOfProcessors);

printf("\n");

printf("\*\*\*\*Состояние памяти\*\*\*\*\*\n");

MEMORYSTATUSEX lpMemStat;

lpMemStat.dwLength = sizeof(MEMORYSTATUSEX);

GlobalMemoryStatusEx(&lpMemStat);

printf("Насколько загружена память: %u%%\n", lpMemStat.dwMemoryLoad);

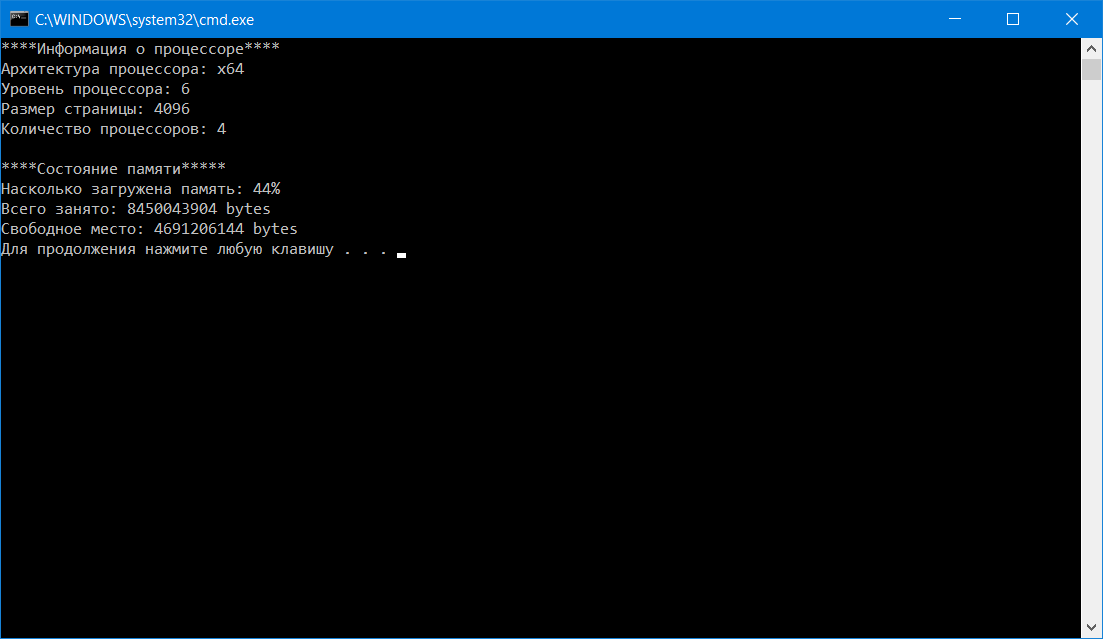
printf("Всего занято: %llu bytes\n", lpMemStat.ullTotalPhys);

printf("Свободное место: %llu bytes\n", lpMemStat.ullAvailPhys);

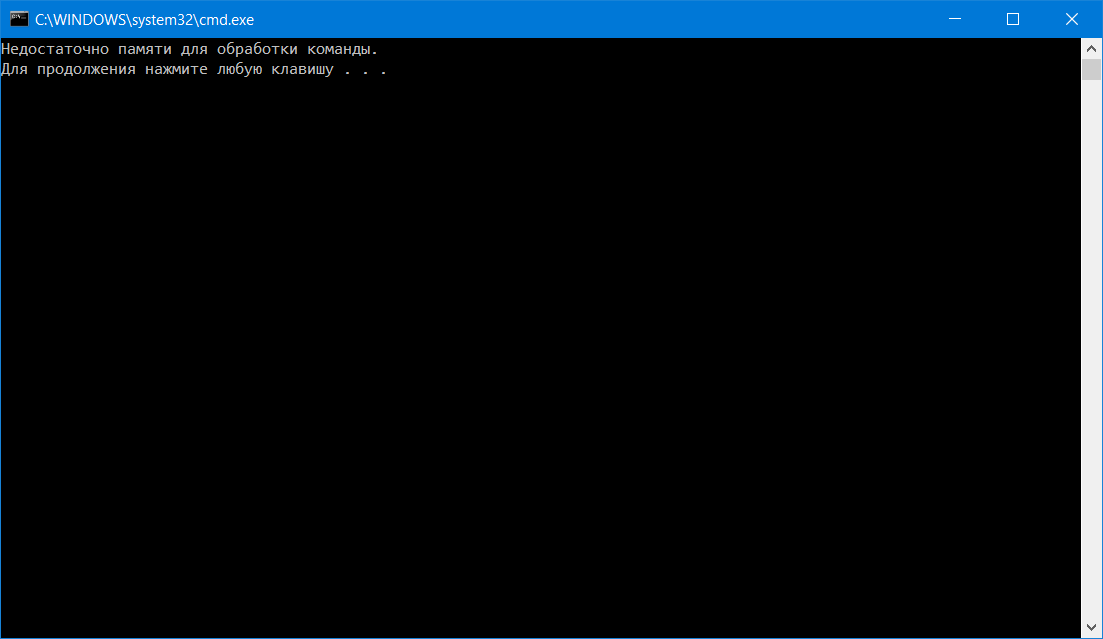
}

**Результат работы программы**

1.Вывод информации об состоянии системы.



2.Генерация ошибки.



**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).

**Текст программы**

В среде Microsoft Visual Studio была написана программа приведенная ниже.

#include "stdafx.h"

void printErrorMessage(HRESULT hResult = NULL);

LPSTR UnicodeToANSI(LPCWSTR src);

LPWSTR ANSIToUnicode(LPCSTR src);

void ANSIToUnicodeFile(LPCTSTR szSourceFile, LPCTSTR szDestFile);

void UnicodeToANSIFile(LPCTSTR szSourceFile, LPCTSTR szDestFile);

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "Rus");

HANDLE hFileSrc = NULL;

if (argc != 4)

{

printf("Неправильные аргументы \n");

}

else if (!lstrcmp(argv[1], L"-a"))

{

ANSIToUnicodeFile(argv[2], argv[3]);

}

else if (!lstrcmp(argv[1], L"-u"))

{

UnicodeToANSIFile(argv[2], argv[3]);

}

else

{

printf("Неправильные аргументы\n");

}

return 0;

}

void ANSIToUnicodeFile(LPCTSTR szSourceFile, LPCTSTR szDestFile)

{

HANDLE hFile;

if (!(hFile = CreateFile(szSourceFile, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL)))

{

printErrorMessage();

return;

}

DWORD dwFileSize = GetFileSize(hFile, NULL);

DWORD dwBytesRead;

CHAR\* szBuf = new CHAR[dwFileSize + 1]; szBuf[dwFileSize] = '\0';

ReadFile(hFile, szBuf, dwFileSize, &dwBytesRead, NULL);

CloseHandle(hFile);

if (!(hFile = CreateFile(szDestFile, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL)))

{

printErrorMessage();

return;

}

WCHAR\* szBufW = ANSIToUnicode(szBuf);

DWORD dwBytesToWrite = lstrlenW(szBufW) \* sizeof(WCHAR);

DWORD dwBytesWritten;

WriteFile(hFile, szBufW, dwBytesToWrite, &dwBytesWritten, NULL);

CloseHandle(hFile);

delete[] szBuf;

}

void UnicodeToANSIFile(LPCTSTR szSourceFile, LPCTSTR szDestFile)

{

HANDLE hFile;

if (!(hFile = CreateFile(szSourceFile, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL)))

{

printErrorMessage();

return;

}

DWORD dwFileSize = GetFileSize(hFile, NULL);

DWORD dwBytesRead;

WCHAR\* szBufW = new WCHAR[dwFileSize / sizeof(WCHAR) + 1]; szBufW[dwFileSize / sizeof(WCHAR)] = 0;

ReadFile(hFile, szBufW, dwFileSize, &dwBytesRead, NULL);

CloseHandle(hFile);

if (!(hFile = CreateFile(szDestFile, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL)))

{

printErrorMessage();

return;

}

CHAR\* szBuf = UnicodeToANSI(szBufW);

DWORD dwBytesToWrite = strlen(szBuf);

DWORD dwBytesWritten;

WriteFile(hFile, szBuf, dwBytesToWrite, &dwBytesWritten, NULL);

CloseHandle(hFile);

delete[] szBuf;

}

LPWSTR ANSIToUnicode(LPCSTR src)

{

if (!src) return 0;

int srcLen = strlen(src);

if (!srcLen)

{

wchar\_t \*w = new wchar\_t[1];

w[0] = 0;

return w;

}

int requiredSize = MultiByteToWideChar(CP\_ACP, 0, src, srcLen, 0, 0);

if (!requiredSize)

{

return 0;

}

wchar\_t\* w = new wchar\_t[requiredSize + 1];

w[requiredSize] = 0;

int retval = MultiByteToWideChar(CP\_ACP, 0, src, srcLen, w, requiredSize);

if (!retval)

{

delete[] w;

return 0;

}

return w;

}

LPSTR UnicodeToANSI(LPCWSTR src)

{

if (!src) return 0;

int srcLen = wcslen(src);

if (!srcLen)

{

char \*x = new char[1];

x[0] = '\0';

return x;

}

int requiredSize = WideCharToMultiByte(CP\_ACP, WC\_COMPOSITECHECK, src, srcLen, 0, 0, 0, 0);

if (!requiredSize)

{

return 0;

}

char \*x = new char[requiredSize + 1];

x[requiredSize] = 0;

int retval = WideCharToMultiByte(CP\_ACP, WC\_COMPOSITECHECK, src, srcLen, x, requiredSize, 0, 0);

if (!retval)

{

delete[] x;

return 0;

}

return x;

}

void printErrorMessage(HRESULT hResult)

{

if (hResult == NULL)

{

hResult = GetLastError();

}

LPTSTR errorText = NULL;

FormatMessage(

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

NULL,

hResult,

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT),

(LPTSTR)&errorText,

0,

NULL);

if (NULL != errorText)

{

wprintf(L"%0s", errorText);

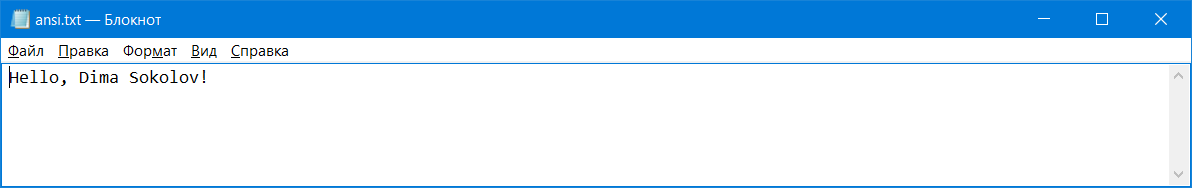
LocalFree(errorText);

}

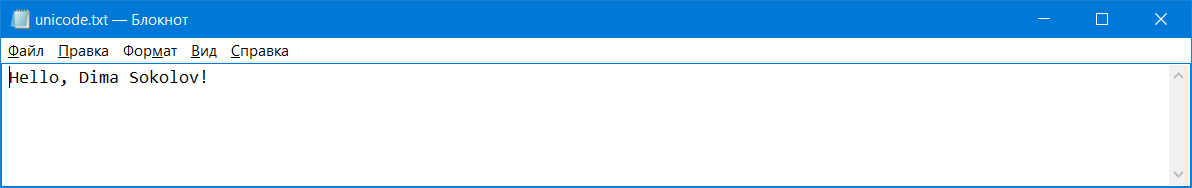
}

**Результат работы программы**

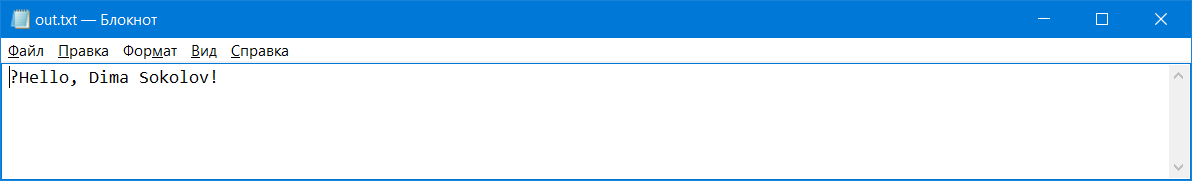
1.Файл с текстом ASCII (ansi).



2.Файл с текстом Юникод (unicode).



3.Выходной файл (out).



4.Результат выполнения.



**Выводы:**

В ходе лабораторной работы я изучил системные вызовы Win32 API, позволяющие получить информацию об ошибке. Изучил функции сбора информации о системе. Также выполнял перекодировку текста из ASCII в Юникод и обратно. В качестве входных данных выступал файл с текстом.