УДК 004.052.4

Мохначёв В.В., Елагин И.А., Теплинский С.В.
Донецкий национальный технический университет
кафедра компьютерной инженерии
E-mail: vlad230596@mail.ru, progressifff@gmail.com

ОПИСАНИЕ И ОБРАБОТКА СИСТЕМНЫХ И ПОЛЬЗОВАТЕЛЬСКИХ ОШИБОК

Мохначёв В.В., Елагин И.А., Теплинский С.В. Описание и обработка системных и пользовательских ошибок. *В статье рассматривается вопрос обработки ошибок, возникающих при выполнении функций windows. Определен основной способ описания собственных ошибок и использование его при программировании с помощью подключения файла-ресурса и динамической библиотеки.*

Введение

Безусловно, обработка ошибок является одним из самых важных процессов разработки приложения, так как программа должна быть понятной пользователю. Пользовательские ошибки возникают повсеместно: ввод неправильных данных, работа не в нужном режиме и т.д. Качественная программа должна определять возникновение ошибки, предоставлять информацию о ней пользователю, и по возможности предлагать варианты решения проблемы. Актуальность данной работы состоит в том, что не смотря на важность данной проблемы, описание её решения встречается редко, к тому же не в полной мере.

Обработка ошибок в функциях Windows

Функция Windows при вызове проверяет переданные ей параметры и начинает выполнять свою работу. Если, например, программист передал недопустимый параметр функция возвращает значение, которое свидетельствует об ошибке. За каждой ошибкой закреплено 32-битный код. Чтобы получить код ошибки используется функция DWORD GetLastError (). Ее вызывают сразу после выполнения функции Windows. Все

системные ошибки Windows описаны в файле winerror.h(обычно в литературе, например, «Рихтер Дж. Windows для профессионалов»[2], рекомендуется использовать именно их).

Анализируя файл winerror.h можно увидеть, что в нем есть две категории ошибок. Первая категория – это коды WIN32 – ошибок, пердставленные в виде десятичного числа, например:

#define ERROR_FILE_NOT_FOUND 2L

L задает тип long, тоесть 32 – х битное значение для компилятора MS Studio 2010.

Вторая категория – это коды ошибок, которые возвращают COM функции, например:

#define E_OUTOFMEMORY _HRESULT_TYPEDEF_(0x8007000EL)

Ошибки из этой категории представлены в виде шестнадцатиричных констант, которые имеют тип HRESULT. Этот тип является одним из средств контроля ошибок в COM и представляет собой 32 – битное число, в котором кодируется результат операции. Нужно отметить, что теже ошибки Win32 также переводятся в HRESULT при помощи специального мкроса HRESULT_FROM_WIN32. Значения двоичного кода этих ошибок разбито на поля как показано в таблице 1.

Таблица 1. Представление кода ошибки в 32 – х битном значении

Биты	31	30 -27	26 - 16	15 - 0
Содержимое	Код стпени «тяжести» (serverity)	Зарезервированные значения	Код подсистемы (facility code)	Код самой ошибки
Значение	0 –успех 1 - ошибка	Определяются Microsoft	Определен Microsoft	Определяется Microsoft или пользователем

Описание собственных ошибок и исользование их при программировании.

В файле winerror.h описано огромное количество ошибок. Но часто возникает ситуация, когда программисту необходимо самому описать ошибку или интерпритировать по-другому ошибку, описанную в winerror.h. Для этого в Windows имеется мощное средство создания собственных сообщений Message

Compilier или MC[3]. Это средство используется для компиляции файлов с расширением .mc. В данном файле определяются сообщения.

Содержимое файла можно разделить на две части: секция - заголовок и секция определений сообщений.

В заголовке определяются имена и идентификаторы языка для определения сообщений в теле файла. В таблице 2 определены следующие значения заголовка

Таблица 2. Синтаксические определения для описания заголовка

Значение	**Описание**
MessageIdTypedef=*type*	Указывается тип кода сообщения
SeverityNames=(*name*=*number*[:*name*])	Указывается важность сообщения
FacilityNames=(*name*=*number*[:*name*])	Указывается тип подсистемы
LanguageNames=(*name*=*number*:*filename*)	Указывается язык, на котором будут определены сообщения
OutputBase=*number*	Указывается в какой системе счисления будет определен код ошибки в десятичной или шестнадцатиричной

На рисунке 1 представлен пример описания заголовка файла сообщений.

```
; // язык сообщений
LanguageNames=(Russian=0x419:MSG_RUS)
LanguageNames=(English=0x409:MSG_ENG)
; // определение категорий сообщений
MessageIdTypedef=DWORD

SeverityNames=(Success=0:STATUS_SEVERITY_SUCCESS
Informational=1:STATUS_SEVERITY_INFORMATIONAL
Warning=2:STATUS_SEVERITY_WARNING
Error=3:STATUS_SEVERITY_ERROR
)

FacilityNames=( System=0x0FF
Application=0xFFF
)
```

Рисунок - 1. Пример описания заголовка файла сообщений

В секции определения сообщений определяются сами сообщения на заданных языках и их вид. В таблице 3 определены следующие значения секции сообщений.

Таблица 3. Синтаксические определения для описания секции сообщений

Значение	**Описание**
MessageId=[*number*|+*number*]	Указывается идентификатор сообщения в шест
Severity=*name*	Указывается описанное ранее значение в заголовке
Facility=*name*	Указывается описанное ранее значение в заголовке
SymbolicName=*name*	Макроопределение сообщения
Language=*name*	Указывается описанное ранее значение в заголовке
message text	Записывается текст на выбранном языке

На рисунке 2 представлен пример описания секции сообщений файла сообщений.

```
MessageId=0x1
Severity=Error
Facility=Runtime
SymbolicName=EMPTY_CMD_LINE_ERROR
Language=English
Command Line is empty.
.
Language=Russian
Недостаточно аргументов командной строки.
.
```

Рисунок - 2. Пример описания секции сообщений файла сообщений

Созданный заполненный файл необходимо скомпилировать. Для этого очень удобно использовать командную строку компилятора Visual Studio. Все, что нужно это добавить заданный файл, в командной строке дать команду mc -A "%(FullPath)", где mc – это компилятор сообщений, флаг -A указывает таблицу кодировки ASCII, FullPath дает полный путь к компилируемому файлу, и указать выводы компиляции %(Filename).rc; %(Filename).h в виде файла ресурса и заголовочного файла. На рисунке 3 представлен пример описанных действий.

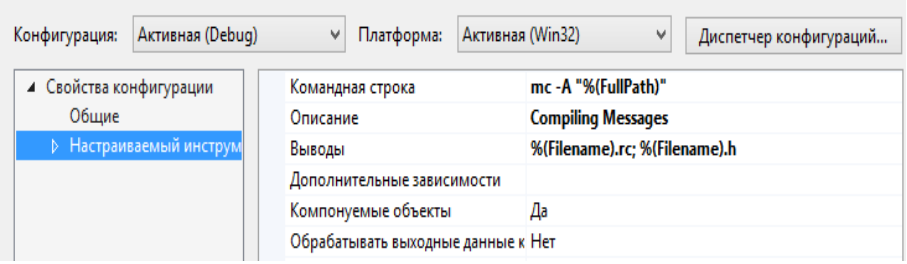

Рисунок - 3. Командная строка для компиляции файла сообщений

В результате компиляции получены заголовочный файл, подобный файлу winerror.h с описанием и индентификаторами в шестнадцатиричной системе счисления ошибок, структура идентификатора ошибки показана в таблице 4, бинарные файлы с

текстом ошибок на заданных языках, файл ресурс с определенными в нем полученными бинарнами файлами и идентификаторами используемых языков. Заголовочный файл и файл ресурс подключаются к проекту и теперь можно использовать описанные ошибки.

Таблица 4. Представление кода описанной ошибки в 32 – х битном значении

Биты	31 - 30	29	28	27 - 16	15 - 0
Содер-жимое	Код стпени «тяжести» (serverity)	Кем определен – Microsoft или пользователем	Зарезерви-рован	Код подсистемы (facility code)	Код самой ошибки
Значе --ние	00 – успех 01 – информация 10 – предупреждение 11 – ошибка	0 – Microsoft 1 - пользователь	Всегда ноль	Опреде-лен Microsoft	Определяется Microsoft или пользователем

Рассмотрим два способа подключения файла с ошибками к программе:

1. Встраивание ресурсного файла(*.res), полученного способом, описанным выше.

2. Создание динамической библиотеки, которая будет хранить описание ошибок.

Первый способ более простой в использовании, а также лучше портируемый, так как описание ошибок будет «вшито» в исполняемый файл. Минусы данного метода в том, что увеличивается размер исполняемого файла, и при использовании несколькими приложениями одних и тех же ошибок, нужно дублировать описание ошибок в каждом файле.

Второй способ позволяет создать библиотеку, которая может быть использована различными приложениями, а храниться она будет только в одном месте. Если есть необходимость создать динамическую библиотеку для описания архитектуры приложения, целесообразно будет добавить к этой библиотеке и файл с ошибками. Что позволит в одном месте хранить и описание ошибок и функции, которые могут их обрабатывать.

Чтобы получить динамическую библиотеку, необходимо создать пустой проект типа DLL библиотека, добавить уже скомилированный файл (*.res). В настройках проекта(Настройки конфигурации->Компоновщик->Дополнительно) установить метку «без точки входа», и скомпилировать проект. После чего можно использовать полученную библиотеку(*.dll).

Разработка

На предыдущих этапах были получены файлы с описаниями собственных ошибок. Для примера реализации работы с ними предлагается рассмотреть небольшое приложение на языке Assembler, которое должно считывать данные из файла(имя файла передаётся в командной строке) и выполнять некоторую их обработку. Собственные ошибки будем хранить в динамической библиотеке, символьное описание кодов ошибок получим путём редактирования заголовочного файла и сохранения его в формате *.inc. Например, строку

#define EMPTY_CMD_LINE_ERROR ((DWORD)0xC0020001L)

заменим на:

EMPTY_CMD_LINE_ERROR equ 0C0020001h

Описание переменных, подключение заголовочных фалов и основная процедура представлены на рисунке 3.

```
include masm32rt.inc
include my_err.inc
        .data
hModule                 dd      0
nameModule              db "MYErrors.dll", 0
err_str                 dw      256 dup(?)
type_error              db "Операция открытия файла.",0
CommandLine     dd      0
argc                    dd      0
```

```
    .code
start:
        invoke SetLastError, 0
        call  CreateConsole
        invoke   LoadLibraryA, offset nameModule
        mov hModule, eax ;подключение модуля с ошибками
        call convertCommandLine
        cmp argc, 1 ;количество аргументов в командной строке
        jg right_console
                invoke SetLastError, EMPTY_CMD_LINE_ERROR;установка ошибки
                lea edi, type_error
                call printError;проверка ошибки
                jmp end_w
        right_console:
                call work
        end_w:
                invoke FreeLibrary, hModule;освобождение файла с ошибками
```

Рисунок - 4. Данные программы и основная процедура

Процедура **SetLastError** загружает в стек ошибок код ошибки(dword значение). Подключение заголовочного файла позволило нам вместо кода ошибки писать его символьный эквивалент. Процедуры **LoadLibrary** и **FreeLibrary** осуществляют соответственно загрузку и освобождение модуля с ошибками из памяти.

Для проверки, произошла ли ошибка, и если да, то какая. Используется процедура, представленная на рисунке 4. Процедура **GetLastError** возвращает из стека ошибок в **eax** код последней погруженной ошибки и смещает указатель стека ошибок на поле одной ошибки. Для формирования кода существуют различные функции, например описанные в [4]. Но Рихтер в своей книге [2] рекомендует использовать процедуру **FormatMessage** как наиболее функциональную и удобную. Описание её аргументов содержится в таблице 5.

```
;edi -> type error[string]
printError      proc
        invoke GetLastError
        test eax, eax
        je no_error
                lea esi,err_str
                invoke FormatMessageA,FORMAT_MESSAGE_FROM_HMODULE + FORMAT_MESSAGE_FROM_SYSTEM
                hModule, eax, 0, esi, 200, 0
                invoke MessageBox, 0, esi, edi,MB_OK + MB_ICONERROR
        no_error:
        ret
printError      endp
```

Рисунок - 5. Процедура для проверки ошибки

Таблица 5. Аргументы для процедуры **FormatMessage**

Параметр	**Значение**
FORMAT_MESSAGE_FROM_HMODULE + FORMAT_MESSAGE_FROM_SYSTEM	Флаги: формирование сообщения из системных сообщений или пользовательской библиотеки
hModule	Дескриптор модуля с описанием своих ошибок
eax	Код ошибки
0	Язык – язык системы(можно указывать код любого языка, который мы описывали в **LanguageNames** в *.mc файле)
esi	Запись сообщения в поле, адрес которого хранит esi
200	Максимальная длина поля сообщения
0	Форматированный вывод не используется

Демонстрация работы программы представлена на рисунке 5.

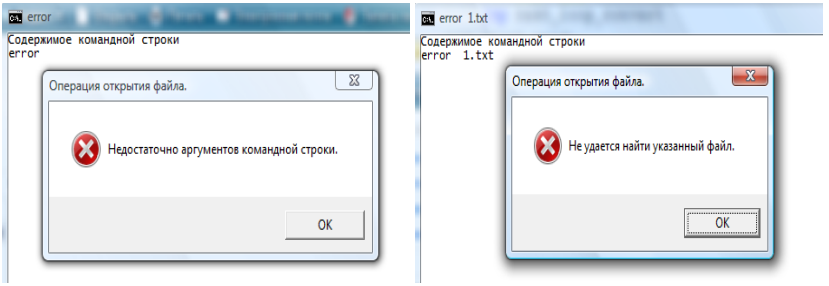

Рисунок - 6. Сообщение об ошибке (слева собственная ошибка, справа системная)

Выводы

В результате работы был рассмотрен вопрос обработки ошибок функций Windows. Также представлен метод описания собственных ошибок и использовании его при написании приложений. Была разработана на языке Assembler программа, демонстрирующая обработку ошибок, описанных в winerror.h, а также промоделирована ситуация возникновения собственной ошибки. Рассмотрены варианты подключения ресурсного файла и динамической библиотеки. Проверка ошибок таким образом, как описано в статье позволяет производить одинаковую обработку как системных, так и пользовательских ошибок.

Список литературы

[1] Финогенов К.Г. Win32. Основы программирования. – 2-е изд., испр. и дополн. – М.:ДИАЛОГ – МИФИ, 2006. – 416с.

[2] Рихтер Дж. Windows для профессионалов: создание эфективных Win32 – приложений с учетом специфии 64 – разрядной версии Windows/ Пер. с англ. – 4-е изд. – Спб.: Питер; М.:Издательство «Русская Редакция»; 2008. – 720 стр.:ил.

[3] Ресурс MSDN. / Интернет-ресурс. - Режим доступа : URL: http://msdn.microsoft.com

[4] Список функций для получения текста ошибки. / Интернет-ресурс. - Режим доступа : URL: http://habrahabr.ru/post/149116/