## Санкт-Петербургский государственный политехнический университет Институт Информационных Технологий и Управления

## Кафедра компьютерных систем и программных технологий

Отчет по расчетной работе № 1

по предмету «Системное программное обеспечение»

**Обработка исключений в ОС Windows**

## Работу выполнил студент гр. 53501/3 Мартынов С. А. Работу принял преподаватель Душутина Е. В.

## Санкт-Петербург 2014

**Оглавление**

[Постановка задачи 3](#_TOC_250013)

[Введение 4](#_TOC_250012)

[Исключения с помощью WinAPI 14](#_TOC_250011)

[Использование GetExceptionCode 24](#_TOC_250010)

[Пользовательская функция-фильтр 30](#_TOC_250009)

[Использование RaiseException 37](#_TOC_250008)

[Необрабатываемые исключения 44](#_TOC_250007)

[Вложенные исключения 52](#_TOC_250006)

[Выход при помощи goto 57](#_TOC_250005)

[Выход при помощи leave 62](#_TOC_250004)

[Преобразование SEH в C++ исключение 67](#_TOC_250003)

[Финальный обработчик finally 72](#_TOC_250002)

[Использование функции AbnormalTermination 76](#_TOC_250001)

[Заключение 82](#_TOC_250000)

# Постановка задачи

## Сгенерировать и обработать исключения с помощью функций WinAPI;

## Получить код исключения с помощью функции GetExceptionCode.

## Использовать эту функции в выражении фильтре;

## Использовать эту функцию в обработчике.

## Создать собственную функцию-фильтр;

## Получить информацию об исключении с помощью функции GetExceptionInformation; сгенерировать исключение с помощью функции RaiseException;

## Использовать функции UnhandledExceptionFilter и SetUnhandledExceptionFilter для необработанных исключений;

## Обработать вложенные исключения;

## Выйти из блока try с помощью оператора goto;

## Выйти из блока try с помощью оператора leave;

## Преобразовать структурное исключение в исключение языка С, используя функцию translator;

## Использовать финальный обработчик finally;

## Проверить корректность выхода из блока try с помощью функции AbnormalTermination в финальном обработчике finally.

## На каждый пункт представить отдельную программу, специфический код, связанный с особенностями генерации заданного исключения структурировать в отдельный элемент (функцию, макрос или иное).

# Введение

## Во время выполнения программы могут возникать ситуации, когда состояние внешних данных, устройств ввода-вывода или компьютерной системы в целом делает дальнейшие вычисления в соответствии с базовым алгоритмом невозможными или бессмысленными. В отсутствие собственного механизма обработки исключений для прикладных программ наиболее общей реакцией на любую исключительную ситуацию является немедленное прекращение выполнения с выдачей пользователю сообщения о характере исключения. Можно сказать, что в подобных случаях единственным и универсальным обработчиком исключений становится операционная система.

## В операционной системе Microsoft Windows, механизм обработки программных и аппа- ратных исключений является SEH (Structured Exception Handling), позволяющий про- граммистам контролировать обработку исключений, а также являющийся отладочным средством.

## В данной работе рассматриваются следующие исключения:

* **EXCEPTION\_FLT\_DIVIDE\_BY\_ZERO** - поток попытался сделать деление на ноль с плавающей точкой;
* **EXCEPTION\_FLT\_OVERFLOW** - переполнение при операции над числами с плавающей точкой.

## Для исследования результатов работы программы будет использоваться тройная система логгирования:

## вывод результатов работы на стандартное устройство вывода (как правило, это экран пользователя, но можно переопределить вывод в файл или на принтер);

## запись протокола работы программы в лог-файл (все лог-файлы находятся в каталоге logs, некоторые приведены по ходу отчёта);

## фиксирование события в системном журнале Windows, который является стандарт- ным способом централизованного хранения информации о важных программных и

## аппаратных событиях (о работе с системным журналом будет рассказано подробнее в одной из задач).

## Каждое событие может быть выгружено из журнала в виде файла. Для этого доступны следующие форматы:

## EVTX (Windows Event Log) – это бинарный файл специфичной структуры;

## XML – форматированный текст;

## TXT – текстовый формат где значения полей разделены символом табуляции;

## CSV – текстовый формат где значения полей разделены запятой.

## Журналы событий представляют собой особые файлы, в которые заносятся сведения о значимых событиях компьютера, например о входе пользователя в систему или об ошибках в приложениях. При возникновении подобных ошибок Windows их регистрирует в соответствующем журнале, который можно прочитать в окне просмотра событий. Сведения в журналах событий могут быть весьма полезны опытным пользователям для устранения неполадок в Windows и других программах.

## Для доступа к просмотру событий журнала требуется нажать кнопку "Пуск выбрать "Па- нель управления "Система и безопасность "Администрирование затем дважды щелкнуть "Просмотр событий"(окно просмотра событий показано на рисунке 1). Выбрать интересую- щий журнал событий можно в левой панели. Для просмотра описания события, нужно дважды кликнуть по нему.

## В окне просмотра событий отслеживаются сведения в нескольких разных журналах. К журналам Windows относятся следующие:

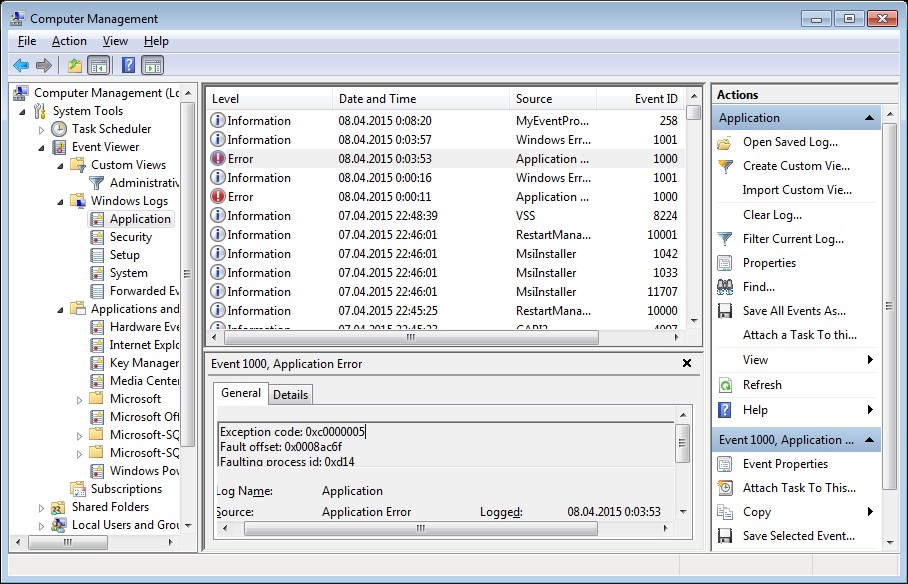
## События приложений (программ). В зависимости от важности события делятся на три категории: ошибка, предупреждение или уведомление. Ошибка указывает

## на серьезную проблему, например потерю данных. Предупреждение указывает на событие, которое в момент записи в журнал не было существенным, но может привести к возникновению проблем в будущем. Информационное событие сообщает об успешной работе приложения, драйвера или службы.

## События, связанные с безопасностью. Такие события называются аудитами и делятся

## на успешные или закончившиеся с ошибкой. Они указывают, например, удалось ли пользователю войти в ОС Windows.

## События установки. Для компьютеров, которые выступают в роли контроллеров домена, здесь отображаются дополнительные журналы.



## Рис. 1: Просмотру событий системного журнала Windows.

## Системные события. Системные события регистрируются Windows и системными службами Windows и подразделяются на ошибки, предупреждения и уведомления.

## Пересылаемые события. Эти события пересылаются в данный журнал другими компьютерами.

## Работа с системным журналом Windows значительно сложнее, чем работа с обычным текстовым файлом т.к. требует компиляции ресурс-файлов.

## Для создания ресурс-файла нужно в меню проекта вызвать добавление нового файла, указать его тип (текстовый файл) и имя (в моем случае это messages.mc). Рисунок 2 показывает процесс создания этого файла.

## Содержимое файла (листинг 1) описывает коды для событий журнала. По представленным комментариям должно быть понятно что происходит: в начале описан язык сообщений (русский) потом две категории сообщений (OVERFLOW\_CATEGORY для событий пе- реполнения при операции над числами с плавающей точкой; OVERFLOW\_CATEGORY для событий деления на ноль) и два определителя сообщений (одно о готовности вызвать исключение, другое о пойманном исключении). Более подробно синтаксис этого файла можно изучить в MSDN https://msdn.microsoft.com/dd996906.aspx.

## Листинг 1: Скрипт генерации ресурсов (src/ExceptionsProcessing/WinAPI/messages.mc)

1. ;// Language
2. Language Names =( Russian =0 x419 : MSG 00419 )
3. Language Names =( English =0 x409 : MSG 00409 )

### 4

5 ;// С ategories

### 6

7 Message Id Typedef = WORD

### 8

1. Message Id =0 x1
2. Symbolic Name = OVERFLOW\_ CATEGORY
3. Language = English
4. An overflow exception category .

13 .

### 14

1. Language = Russian
2. События переполнения

17 .

### 18

1. Message Id =0 x2
2. Symbolic Name = ZERODIVIDE\_ CATEGORY
3. Language = English
4. A division by zero exception category .

23 .

### 24

1. Language = Russian
2. События деления на 0

27 .

### 28

29 ;// Determiners

### 30

31 Message Id Typedef = DWORD

### 32

1. Message Id =0 x100
2. Symbolic Name = READY\_ FOR\_ EXCEPTION
3. Language = English
4. Ready for generate exception .

37 .

### 38

1. Language = Russian
2. Готовность приложения сгенерировать исключительное событие.

41 .

### 42

1. Message Id =0 x101
2. Symbolic Name = CAUGHT\_ EXCEPRION

45

46

47

48

49

50

51

Language = English

Exclusive event happened .

.

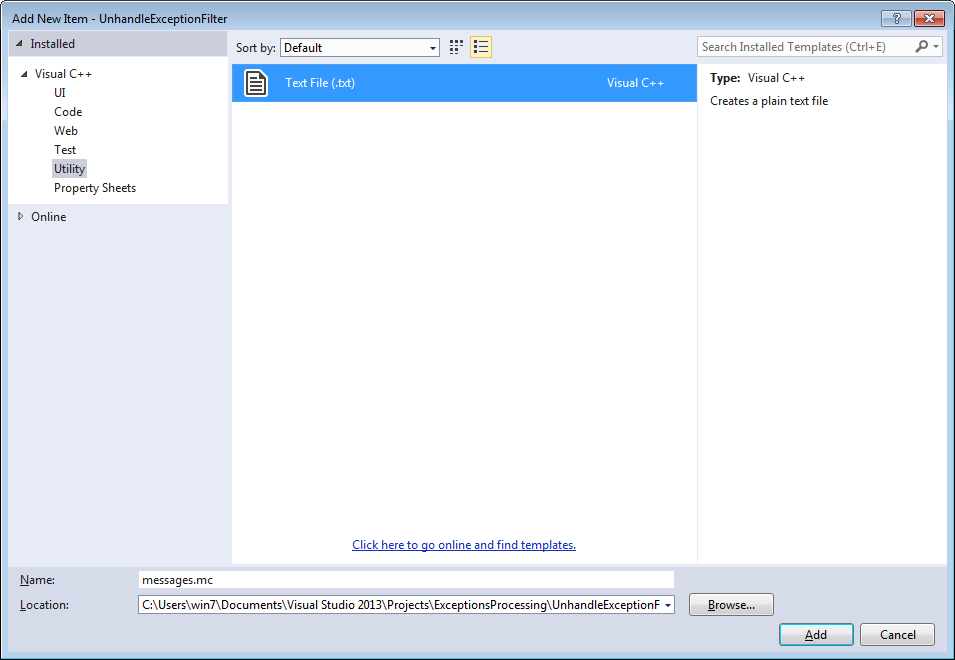
Language = Russian

Произошло ( и поймано) исключительное событие.

.

## Имея этот скрипт, можно перейти в папку, где он находится, и выполнить команду

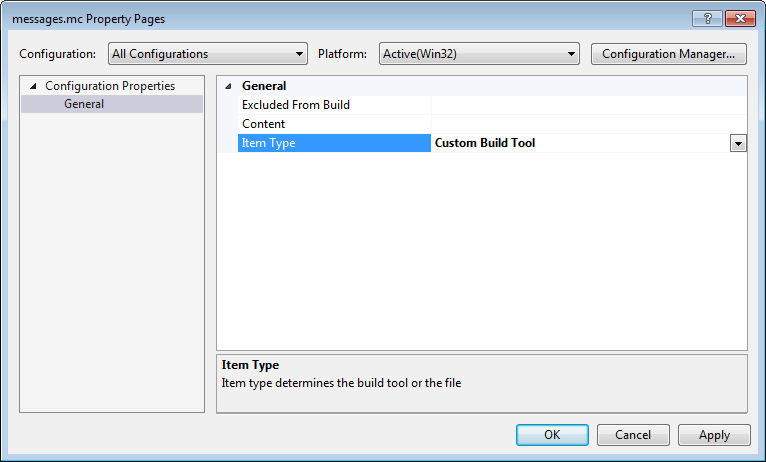
## mc -U messages.mc



## Рис. 2: Создание скрипта для генерации ресурсов в проекте.

## Но можно настроить среду разработки так, чтобы файл компилировался автоматически во время сборки проекта. Для этого нужно вызвать свойства файла messages.mc и в поле "Типа элемента"выбрать "Настраиваемый инструмент построения"(на рисунке 3 этот выбор подсвечен жирным текстом).

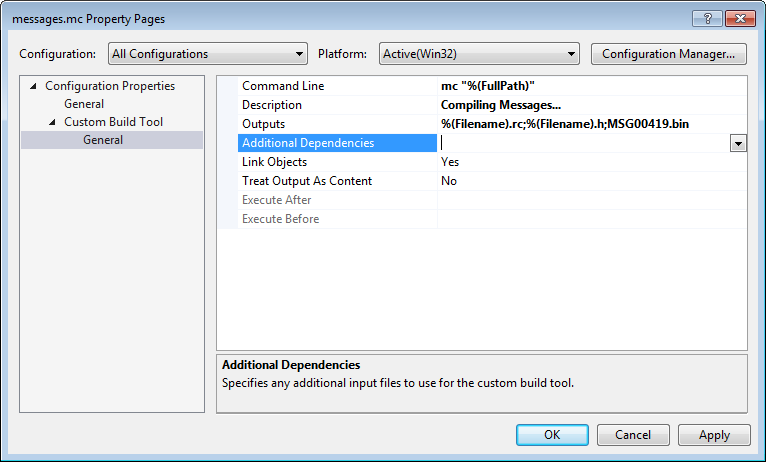
## После того, как будет нажата кнопка "применить в левой панели появилась группа "На- страиваемый инструмент построения где нужно выбрать следующие параметры (смотри рисунок 4):



## Рис. 3: Выбор типа элемента для файла messages.mc

## Командная строка: mc "%(FullPath)" Описание: Compiling Messages...

## Выводы: %(Filename).rc;%(Filename).h;MSG00419.bin



## Рис. 4: Настройки исполнения скрипта генерации ресурсов

## Теперь, при сборке проекта, ресурсы будут сгенерированы автоматически:

## message.h – заголовочный файл ресурсов (см. листинг 2), его необходимо добавить в проект.

## message.rc – файл с описанием ресурсов (моя бесплатная версия Microsoft Visual

## Studio Express не позволяет редактировать этот файл прямо из среды разработки), необходимо добавить в проект;

## message.bin – бинарный файл файл ресурсов.

## После того, как заголовочный файл (см. листинг 2) будет добавлен в проект, можно будет пользоваться определёнными в нём константами.

## Листинг 2: Заголовочный файл для работы с ресурсами (src/ExceptionsProcessing/WinAPI/messages.h)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*//*

*Language*

*Tategories*

*Values*

*are*

*32 bit values laid out as follows:*

*3 3 2*

*1 0 9*

*2 2*

*8 7*

*2 2 2 2 2 2 2*

*6 5 4 3 2 1 0*

*1 1 1 1 1 1*

*9 8 7 6 5 4*

*1 1 1 1*

*3 2 1 0 9 8 7 6 5 4 3 2 1 0*

*+---+-+-+-----------------------+-------------------------------+*

*| Sev| C| R|*

*Facility*

*|*

*Code*

*|*

*+---+-+-+-----------------------+-------------------------------+*

*where*

*Sev - is the severity code*

1. *- Success*
2. *- Informational*
3. *- Warning*
4. *- Error*

*C - is the Customer code flag*

*R - is a reserved bit*

*Facility - is the facility code*

*Code - is the facility ’s status code*

30 *// Define the facility codes*

31 *//*

### 32

### 33

34 *//*

35 *// Define the severity codes*

36 *//*

### 37

### 38

39 *//*

40 *// Message Id : OVERFLOW\_ CATEGORY*

41 *//*

42 *// Message Text :*

43 *//*

44 *// An overflow exception category .*

45 *//*

46 # define OVERFLOW\_ CATEGORY (( WORD ) 0 x 00000001 L )

### 47

48 *//*

49 *// Message Id : ZERODIVIDE\_ CATEGORY*

50 *//*

51 *// Message Text :*

52 *//*

53 *// A division by zero exception category .*

54 *//*

55 # define ZERODIVIDE\_ CATEGORY (( WORD ) 0 x 00000002 L )

### 56

57 *// Determiners*

58 *//*

59 *// Message Id : READY\_ FOR\_ EXCEPTION*

60 *//*

61 *// Message Text :*

62 *//*

63 *// Ready for generate exception .*

64 *//*

65 # define READY\_ FOR\_ EXCEPTION (( DWORD ) 0 x 00000100 L )

### 66

67 *//*

68 *// Message Id : CAUGHT\_ EXCEPRION*

69 *//*

70 *// Message Text :*

71 *//*

72 *// Exclusive event happened .*

73 *//*

74 # define CAUGHT\_ EXCEPRION (( DWORD ) 0 x 00000101 L )

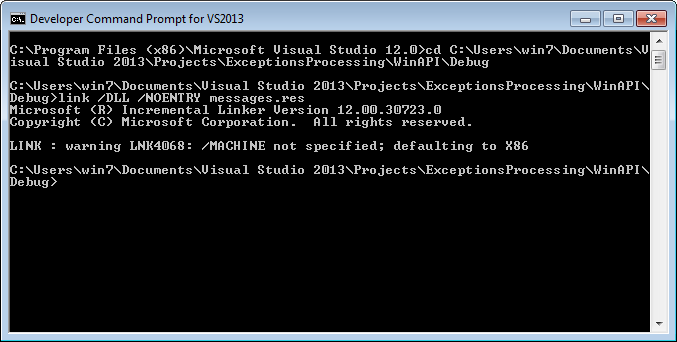
## После этого с системным журналом уже можно работать, но каждое событие будет начи- наться с записи:

*The description for Event ID 258 from source MyEventProvider cannot be found. Either the component that raises this event is not installed on your local computer or the installation is corrupted. You can install or repair the component on the local computer.* (Не удается найти описание для идентификатора события 258 из источника MyEventProvider. Вызывающий данное событие компонент не установлен на этом локальном компьютере или поврежден. Установите или восстановите компонент на локальном компьютере.)

## Для исправления этой ситуации нужно сгенерировать библиотеку с ресурсами и зареги- стрировать её в системе (см. рис. 5). Это делается при помощи командной строки Visual Studio (Developer Command Prompt for VS2013; не путать с интерпретатором CMD!), в котором делается переход в папку, содержащую messages.res (это может быть папка debug) и выполняется команду

## link /DLL /NOENTRY messages.res

## После выполнения этой команды будет создан файл messages.dll.



## Рис. 5: Генерация библиотеки ресурсов

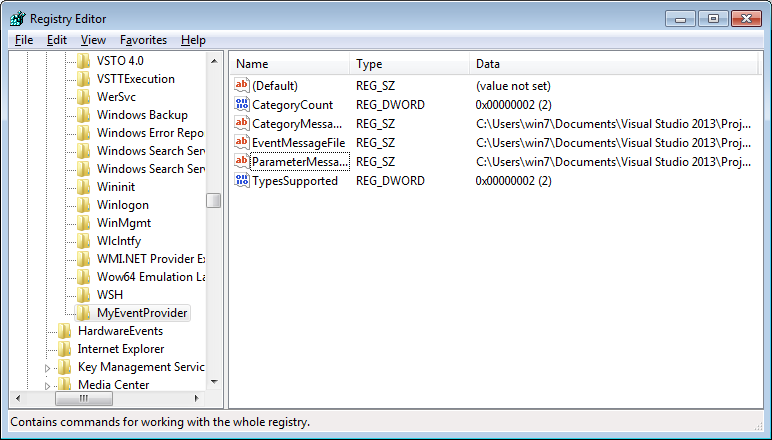
## При внедрении программы необходимо будет зарегистрировать resources.dll в реестре. Для этого потребуется создать ключ MyEventProvider в ветке реестра HKEY\_LOCAL\_MACHINE*∖* SYSTEM*∖*CurrentControlSet*∖*services*∖*eventlog*∖*Application и выставить параметры, перечис-

## ленные в таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Значение | Примечание |
| CategoryCount | REG\_DWORD | 0x00000002 | количество категорий сообщений |
| CategoryMessageFile | REG\_SZ | path*∖*resources.dll | путь до DLL |
| EventMessageFile | REG\_SZ | path*∖*resources.dll | путь до DLL |
| ParameterMessageFile | REG\_SZ | path*∖*resources.dll | путь до DLL |
| TypesSupported | REG\_DWORD | 0x00000002 | количество типов сообщений |

## Таблица 1: Значения для заполнения реестра

## Файл с библиотекой стоит перенести в более подходящее место, но общий вид системного реестра должен выглядеть как рисунок 6.



## Рис. 6: Редактирование системного реестра

## После этого события в системном журнале должны отображаются нормально.

## Все результаты, представленные в данном отчёте получены с использованием Microsoft Windows 7 Ultimate Service Pack 1 64-bit (build 7601). Для разработки использовалась Microsoft Visual Studio Express 2013 for Windows Desktops (Version 12.0.30723.00 Update 3). В качестве отладчика использовался Microsoft WinDbg (release 6.3.9600.16384), работа с которым будет подробнее рассмотрена на одной из задач.

# Исключения с помощью WinAPI

## Задачей этого раздела является генерирование и обработка исключений с помощью функ- ций WinAPI.

## В листинге 3 показана работа с исключениями. В зависимости от параметра, передаваемого при запуске, вызывается либо исключение деления на ноль, либо переполнение разрядной сетки при работе с типом float. Особо стоит обратить внимание на две вещи: изначально, все ошибки типа float маскируются, и для получения исключений нужно от этого маскирования избавиться (см. стр. 68-70); кроме того, операции с плавающими точками выполняются асинхронно, и нужно на этапе компиляции отключить расширения векторизации.

## В 74-й строке используется квалификатор volatile, это помогает обмануть статический анализатор среды разработки (visual studio), который честно сигнализирует о явной ошибке (делении на ноль) и не позволяет собрать программу.

## Листинг 3: Генерация и обработка исключения с помощью функций WinAPI (src/ExceptionsProcessing/WinAPI/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

*/\**

*Task 1.*

*Generate and handle exceptions using the Win API functions ;*

*\*/*

*//*

*//*

*//*

*IMPORTANT : Don ’ t forget to disable Enhanced Instructions !!!*

*Properties -> Configuration Properties -> C/ C++ -> Code Generation -> Enable Enhanced Instruction Set = No Enhanced Instructions (/ arch: IA32 )*

# include

# include

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cstring >

< cfloat >

< cmath >

< excpt . h>

< windows . h>

< time . h>

18 # include " messages . h"

### 19

1. *// log*
2. FILE \* logfile ;
3. HANDLE eventlog ;

### 23

1. void usage ( const \_ TCHAR \* prog );
2. void initlog ( const \_ TCHAR \* prog );
3. void closelog ();
4. void writelog ( \_ TCHAR \* format , ...) ;
5. void syslog ( WORD category , WORD identifier , LPWSTR message );

### 29

1. *// Task switcher*
2. enum {
3. DIVIDE\_ BY\_ ZERO ,
4. FLT\_ OVERFLOW
5. } task ;

### 35

1. *// Defines the entry point for the console application .*
2. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
3. *// Init log*
4. initlog ( argv [0]) ;
5. eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 41

1. *// Check parameters number*
2. if ( argc != 2) {
3. \_ tprintf (\_T(" Too few parameters .\ n\ n"));
4. writelog (\_T(" Too few parameters ."));
5. usage ( argv [0]) ;
6. closelog ();
7. exit (1) ;

49 }

### 50

51 *// Set task*

52 if (! \_ tcscmp (\_T("- d"), argv [1]) ) {

1. task = DIVIDE\_ BY\_ ZERO ;
2. writelog (\_T(" Task : DIVIDE\_ BY\_ ZERO exception ."));

55 }

1. else if (! \_ tcscmp (\_T("-o"), argv [1]) ) {
2. task = FLT\_ OVERFLOW ;
3. writelog (\_T(" Task : FLT\_ OVERFLOW exception ."));

59 }

1. else {
2. \_ tprintf (\_T(" Can ’t parse parameters .\ n\ n"));
3. writelog (\_T(" Can ’t parse parameters ."));
4. usage ( argv [0]) ;
5. closelog ();
6. exit (1) ;

66 }

### 67

1. *// Floating point exceptions are masked by default .*
2. \_ clearfp ();
3. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 71

1. *// Set exception*
2. \_\_ try {
3. volatile float tmp = 0;
4. switch ( task ) {
5. case DIVIDE\_ BY\_ ZERO :
6. writelog (\_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
7. syslog ( ZERODIVIDE\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
8. \_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
9. tmp = 1 / tmp ;
10. writelog (\_T(" DIVIDE\_ BY\_ ZERO exception is generated ."));
11. break ;
12. case FLT\_ OVERFLOW :
13. *// Note: floating point execution happens asynchronously .*
14. *// So , the exception will not be handled until the next floating*
15. *// point instruction .*
16. writelog (\_T(" Ready for generate FLT\_ OVERFLOW exception ."));
17. syslog ( OVERFLOW\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
18. \_T(" Ready for generate FLT\_ OVERFLOW exception ."));
19. tmp = pow ( FLT\_MAX , 3);
20. writelog (\_T(" Task : FLT\_ OVERFLOW exception is generated ."));
21. break ;
22. default :
23. break ;

95 }

96 }

1. \_\_ except ( EXCEPTION\_ EXECUTE\_ HANDLER ) {
2. \_ tprintf (\_T(" Well , it looks like we caught something ."));
3. writelog (\_T(" Exception is caught ."));
4. syslog ( OVERFLOW\_ CATEGORY , CAUGHT\_ EXCEPRION ,
5. \_T(" Well , it looks like we caught something ."));

### 102 }

### 103

1. closelog ();
2. Close Handle ( eventlog );
3. exit (0) ;

### 107 }

### 108

1. *// Usage manual*
2. void usage ( const \_ TCHAR \* prog ) {
3. \_ tprintf (\_T(" Usage : \ n"));
4. \_ tprintf (\_T("\ t% s -d\ n"), prog );
5. \_ tprintf (\_T("\ t\ t\ t for exception float divide by zero ,\ n"));
6. \_ tprintf (\_T("\ t% s -o\ n"), prog );
7. \_ tprintf (\_T("\ t\ t\ t for exception float overflow .\ n"));

### 116 }

### 117

1. void initlog ( const \_ TCHAR \* prog ) {
2. \_ TCHAR logname [ 255 ];
3. wcscpy\_ s ( logname , prog );

### 121

1. *// replace extension*
2. \_ TCHAR \* extension ;
3. extension = wcsstr ( logname , \_T(". exe "));
4. wcsncpy\_ s ( extension , 5 , \_T(". log "), 4);

### 126

1. *// Try to open log file for append*
2. if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {
3. \_ wperror (\_T(" The following error occurred "));
4. \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname );
5. exit (1) ;

### 132 }

### 133

134 writelog (\_T("% s is starting ."), prog );

### 135 }

### 136

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

### 140 }

### 141

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;

### 145

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 148

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

|  |  |  |
| --- | --- | --- |
| 153 |  | |
| 154 |  | *// Convert to normal representation .* |
| 155 |  | swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday , |
| 156 |  | newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour , |
| 157  158 |  | newtime . tm\_min , newtime . tm\_ sec ); |
| 159 |  | *// Write date and time* |
| 160 |  | fwprintf ( logfile , \_T("% s"), buf ); |
| 161  162 |  | *// Write all params*  va\_ start ( ap , format ); |
| 163 |  | \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap); |
| 164 |  | fwprintf ( logfile , \_T("% s"), buf ); |
| 165 |  | va\_ end ( ap); |
| 166 |  | *// New sting* |
| 167 |  | fwprintf ( logfile , \_T("\ n")); |
| 168 | } |  |
| 169 |  |  |
| 170 | vo | id syslog ( WORD category , WORD identifier , LPWSTR message ) { |
| 171 |  | LPWSTR p Messages [1] = { message }; |
| 172 |  |  |
| 173 |  | if (! Report Event ( |
| 174 |  | eventlog , *// event log handle* |

## Произведём три запуска, первый раз без аргументом (для демонстрации зависимости ис- ключения от передаваемого аргумента), второй раз с аргументом "d"(DIVIDE\_BY\_ZERO) и третий раз с аргументом "o"(FLT\_OVERFLOW). Как видно на рисунке 7, первый запуск не дал результатов, второй и третий привёл к исключительной ситуации.

175

176

177

178

179

180

181

182

183

184

185

EVENTLOG\_ INFORMATION\_ TYPE , *// event type*

category ,

identifier , NULL ,

1 ,

0 ,

*//*

*//*

*//*

*// event category*

*// event identifier user security identifier*

*number of substitution strings*

*data size*

( LPCWSTR \*) pMessages ,

*// pointer to strings*

NULL )) {

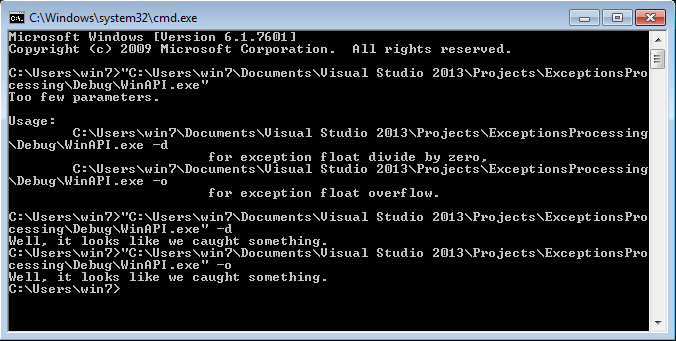
*// pointer to binary data buffer*

writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

}

}

## Во время второго и третьего запуска, управление сразу после исключения с 80-й строки (где происходит исключительная ситуация), передаётся на 97-ю (в которой ожидается исключение), а потом на 104-ю (т.е. обратной передачи управления не происходит), где происходит закрытие используемых дескрипторов и завершение работы. Запись из 81-й и



## Рис. 7: Запуск программы, генерирующей исключения средствами WinAPI.

## 91-й строк в листинге 4 (лог-файл) отсутствуют, т.к. управление до этих строк не дошло.

## В рамках данной задачи мы не разбираем, какое именно исключение произошло, поэто- му в системный журнал все пойманные события помечаются как события из группы переполнения.

## Листинг 4: Генерация и обработка исключения с помощью функций WinAPI

1 [ 6 / 2 / 2015 15 : 32 : 58 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Exceptions Processing \ Debug \ Win API . exe is starting .

Exceptions Processing \ Debug \ Win API . exe is starting .

6

7

8

9

10

11

[ 6 / 2 / 2015

[ 6 / 2 / 2015 [ 6 / 2 / 2015 [ 6 / 2 / 2015

15 : 33 : 5 ] Task : DIVIDE\_ BY\_ ZERO exception .

15 : 33 : 5 ] Ready for generate DIVIDE\_ BY\_ ZERO exception . 15 : 33 : 5 ] Exception is caught .

15 : 33 : 5 ] Shutting down .

[ 6 / 2 / 2015

15 : 33 : 10 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Exceptions Processing \ Debug \ Win API . exe is starting .

1. [ 6 / 2 / 2015 15 : 33 : 10 ] Task : FLT\_ OVERFLOW exception .
2. [ 6 / 2 / 2015 15 : 33 : 10 ] Ready for generate FLT\_ OVERFLOW exception .

14 [ 6 / 2 / 2015 15 : 33 : 10 ] Exception is caught .

15 [ 6 / 2 / 2015 15 : 33 : 10 ] Shutting down .

|  |  |  |
| --- | --- | --- |
| 2 | [ 6 / 2 / 2015 | 15 : 32 : 58 ] Too few parameters . |
| 3 | [ 6 / 2 / 2015 | 15 : 32 : 58 ] Shutting down . |
| 4 |  |  |
| 5 | [ 6 / 2 / 2015 | 15 : 33 : 5 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \ |

## Из примечательного в этом коде ещё строка 173. В ней показан вызов команды ReportEvent, которая непосредственно передаёт строку в системный журнал. Синтаксис команды следу- ющий:

## BOOL ReportEvent(

## \_In\_ HANDLE hEventLog,

## \_In\_ WORD wType,

## \_In\_ WORD wCategory,

## \_In\_ DWORD dwEventID,

## \_In\_ PSID lpUserSid,

## \_In\_ WORD wNumStrings,

## \_In\_ DWORD dwDataSize,

## \_In\_ LPCTSTR \*lpStrings,

## \_In\_ LPVOID lpRawData

## );

## Значения полей следующие:

## hEventLog – описатель логера (инициализирован в строке 22);

## wType – тип события (ошибка, предупреждение, успех...);

## wCategory – категория события (определяется пользовательским кодом);

## dwEventID – определитель события (определяется пользовательским кодом);

## lpUserSid – указатель на идентификатор безопасности пользователя (может быть NULL)ж

## wNumStrings – количество строк в сообщении события;

## dwDataSize – количество байт в прилагаемом бинарном участке;

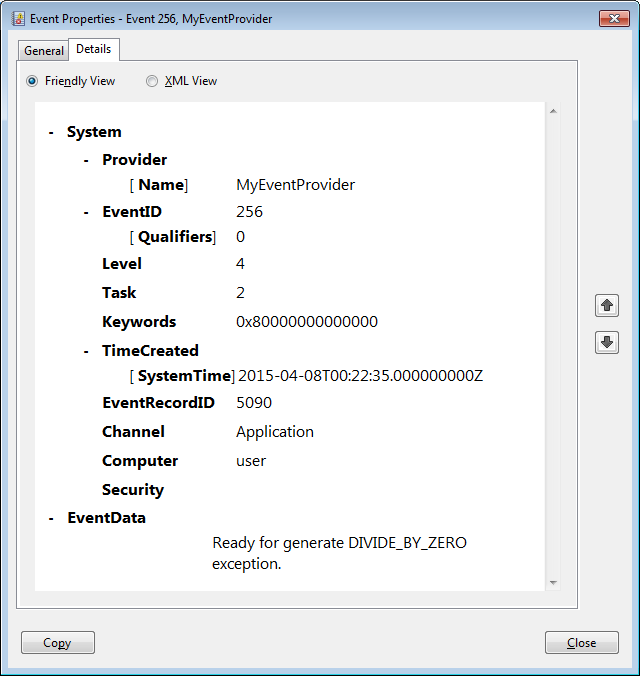
## lpStrings – указатель на массив строк;

## lpRawData – указатель на бинарный участок.

## В случае успеха, функция возвращает не нулевой результат. Увидеть зафиксированное событие можно на рисунке 8. Информация события представляет меньше полезной ин- формации, чем лог-файл, так что далее все события будут сохраняться в папку logs, но в отчёте приводиться не будут.

## Теперь можно перейти к отладке кода при помощи WinDbg.

## WinDbg — позволяет отлаживать 32/64 битные приложения пользовательского уровня, драйвера, может быть использован для анализа аварийных дампов памяти, WinDbg поддер-

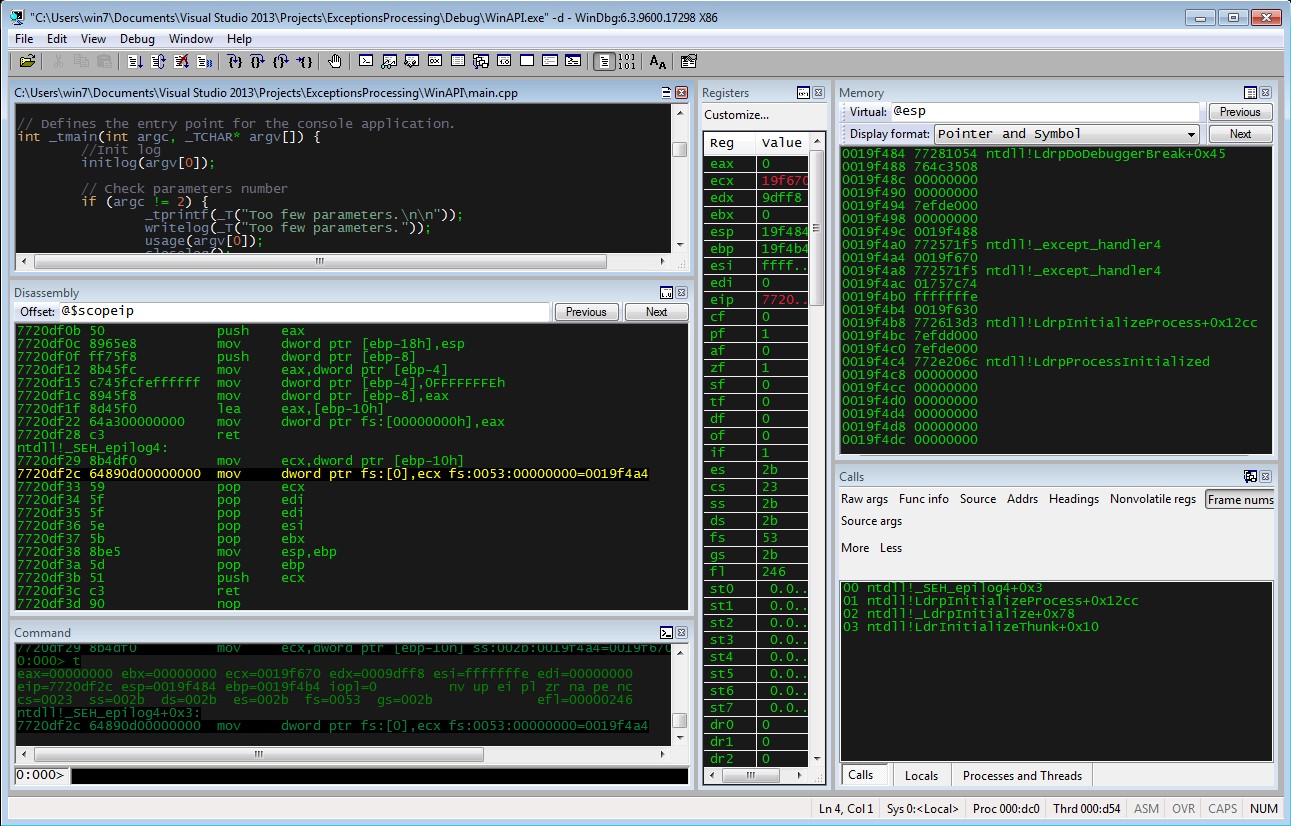


## Рис. 8: Просмотр события в "дружелюбном"виде (альтернативой которому является XML).

## живает автоматическую загрузку отладочных символов, имеется встроенный скриптовый язык для автоматизации процесса отладки, а самое главное распространяется корпорацией Microsoft совершенно бесплатно.

## В отличии от OllyDbg, WinDbg при первом запуске имеет достаточно неприятный интер- фейс и требуется довольно много усилий и времени для подготовки этого инструмента к комфортной работе. На рисунке 9 можно увидеть интерфейс WinDbg и запущенную программу WinAPI.

## Экран разделён на три участка. В левой части представлены три окна: окно исходного кода (там отображаются даже комментарии на русском языке), под ним окно с ассемблерным кодом (имеется возможность устанавливать точки основа как в кода не С, так и в ассем- блерном коде), а под ним диалоговое окно, в котором отображается результат выполнения запросов пользователя. Центральное окно представляет таблицу значений регистров цен-



## Рис. 9: Запуск WinAPI.exe под отладчиком WinDbg.

## трального процессора. В правой части снова три окна: два верхних показывают состояние памяти (но можно выбрать представление, допустим в одном случае память выглядит как набор байтов, а в другом как юникод - это удобно для быстрого переключения между различными сегментами), а под ними окно стека.

## Контроль исполнения:

## g – продолжить выполнение.

## p – шаг через функцию.

## t – шаг внутрь функции.

## pa addr – шаг в адрес.

## pc – шаг в следующий вызов.

## pt – шаг к следующему возврату.

## pct – шаг к следующему вызову или возврату.

## Точки останова:

## bp – установка точки останова, например bp nt!NtCreateFile.

## bl – список точек останова.

## bd – <число> убрать точку останову под номером.

## bc – <число> очистить точку останова под номером.

## ba – точка останова на доступ.

## be – точка останова на исполнение.

## bw – точка останова на запись.

## sxe ld:kernel32 – точка останова на загрузке DLL модуля.

## Работа с дампом (в данной работе не требуется, но для полноты картины):

## d <адрес> – дамп памяти по адресу (b-byte;w-word;d-dword).

## dd <регистр> – дамп содержимого регистра.

## ddp <адрес> – дамп содержимого по адресу.

## u <адрес> – дизассемблировать по адресу.

## Передачу управления можно видеть и по средствам отладчика. В правом нижнем углу показан стек. В данном случае глубина стека не достаточно большая для наглядного изучения поиска обработчика, но вызов обработчика на нём виден.

## Благородя тому, что исключение было обработано, оно не дошло до уровня операционной системы, и не было отражено в системном журнале как ошибка.

# Использование GetExceptionCode

## Функция GetExceptionCode позволяет получить код исключения, которое было сгенери- ровано в процессе работы программы (смотри листинг 5). В первом случае (строка 86) она участвует в сравнении с макро-константной EXCEPTION\_FLT\_DIVIDE\_BY\_ZERO для определения подходящего обработчика для исключительного события. Во-втором случае (строка 109) она используется уже внутри обработчика, позволяя определить, что исключение вызвано переполнением при операции с типом float.

## Листинг 5: Получение кода исключения с помощью функции GetExceptionCode (src/ExceptionsProcessing/GetExceptionCode/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

*/\**

*Task 2.*

*Get the exceptions code using the Get Exception Code gunction :*

* *Use this function in the filter expression ;*
* *Use this function in the handler.*

*\*/*

*//*

*//*

*//*

*IMPORTANT : Don ’ t forget to disable Enhanced Instructions !!!*

*Properties -> Configuration Properties -> C/ C++ -> Code Generation -> Enable Enhanced Instruction Set = No Enhanced Instructions (/ arch: IA32 )*

# include

# include

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cstring >

< cfloat >

< cmath >

< excpt . h>

< windows . h>

< time . h>

# include

" messages . h"

*// log*

FILE \* logfile ; HANDLE eventlog ;

1. void usage ( const \_ TCHAR \* prog );
2. void initlog ( const \_ TCHAR \* prog );
3. void closelog ();
4. void writelog ( \_ TCHAR \* format , ...) ;
5. void syslog ( WORD category , WORD identifier , LPWSTR message );

### 31

1. *// Task switcher*
2. enum {
3. DIVIDE\_ BY\_ ZERO ,
4. FLT\_ OVERFLOW
5. } task ;

### 37

1. *// Defines the entry point for the console application .*
2. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
3. *// Init log*
4. initlog ( argv [0]) ;
5. eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 43

1. *// Check parameters number*
2. if ( argc != 2) {
3. \_ tprintf (\_T(" Too few parameters .\ n\ n"));
4. writelog (\_T(" Too few parameters ."));
5. usage ( argv [0]) ;
6. closelog ();
7. exit (1) ;

51 }

### 52

53 *// Set task*

54 if (! \_ tcscmp (\_T("- d"), argv [1]) ) {

1. task = DIVIDE\_ BY\_ ZERO ;
2. writelog (\_T(" Task : DIVIDE\_ BY\_ ZERO exception ."));

57 }

1. else if (! \_ tcscmp (\_T("-o"), argv [1]) ) {
2. task = FLT\_ OVERFLOW ;
3. writelog (\_T(" Task : FLT\_ OVERFLOW exception ."));

61 }

1. else {
2. \_ tprintf (\_T(" Can ’t parse parameters .\ n\ n"));
3. writelog (\_T(" Can ’t parse parameters ."));
4. usage ( argv [0]) ;
5. closelog ();
6. exit (1) ;

68 }

### 69

70 *// Floating point exceptions are masked by default .*

1. \_ clearfp ();
2. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 73

1. *// Set exception*
2. volatile float tmp = 0;
3. switch ( task ) {
4. case DIVIDE\_ BY\_ ZERO :
5. \_\_ try {
6. writelog (\_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
7. syslog ( ZERODIVIDE\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
8. \_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
9. tmp = 1 / tmp ;
10. writelog (\_T(" DIVIDE\_ BY\_ ZERO exception is generated ."));

84 }

1. *// Use Get Exception Code () function in the filter expression ;*
2. \_\_ except (( Get Exception Code () == EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ) ?
3. EXCEPTION\_ EXECUTE\_ HANDLER :
4. EXCEPTION\_ CONTINUE\_ SEARCH )

89 {

1. \_ tprintf (\_T(" Caught exception is: EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO "));
2. writelog (\_T(" Caught exception is: EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO "));
3. syslog ( ZERODIVIDE\_ CATEGORY , CAUGHT\_ EXCEPRION ,
4. \_T(" Caught exception is: EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ."));

94 }

1. break ;
2. case FLT\_ OVERFLOW :
3. \_\_ try {
4. *// Note: floating point execution happens asynchronously .*
5. *// So , the exception will not be handled until the next*
6. *// floating point instruction .*
7. writelog (\_T(" Ready for generate FLT\_ OVERFLOW exception ."));
8. syslog ( OVERFLOW\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
9. \_T(" Ready for generate FLT\_ OVERFLOW exception ."));
10. tmp = pow ( FLT\_MAX , 3);
11. writelog (\_T(" Task : FLT\_ OVERFLOW exception is generated ."));

### 106 }

1. *// Use Get Exception Code () function in the handler .*
2. \_\_ except ( EXCEPTION\_ EXECUTE\_ HANDLER ) {
3. if ( Get Exception Code () == EXCEPTION\_ FLT\_ OVERFLOW ) {
4. writelog (\_T(" Caught exception is: EXCEPTION\_ FLT\_ OVERFLOW "));
5. \_ tprintf (\_T(" Caught exception is: EXCEPTION\_ FLT\_ OVERFLOW "));
6. syslog ( OVERFLOW\_ CATEGORY , CAUGHT\_ EXCEPRION ,
7. \_T(" Caught exception is: EXCEPTION\_ FLT\_ OVERFLOW ."));

### 114 }

115 else {

|  |  |  |
| --- | --- | --- |
| 116 |  | writelog (\_T(" UNKNOWN exception : % x\ n"), Get Exception Code ()); |
| 117 |  | \_ tprintf (\_T(" UNKNOWN exception : % x\ n"), Get Exception Code ()); |
| 118 |  | } |
| 119 |  | } |
| 120 |  | break ; |
| 121 |  | default : |
| 122 |  | break ; |
| 123 |  | } |
| 124 |  | closelog (); |
| 125 |  | Close Handle ( eventlog ); |
| 126 |  | exit (0) ; |
| 127 | } |  |
| 128 |  |  |
| 129 | *//* | *Usage manual* |
| 130 | vo | id usage ( const \_ TCHAR \* prog ) { |
| 131 |  | \_ tprintf (\_T(" Usage : \ n")); |
| 132 |  | \_ tprintf (\_T("\ t% s -d\ n"), prog ); |
| 133 |  | \_ tprintf (\_T("\ t\ t\ t for exception float divide by zero ,\ n")); |
| 134 |  | \_ tprintf (\_T("\ t% s -o\ n"), prog ); |
| 135 |  | \_ tprintf (\_T("\ t\ t\ t for exception float overflow .\ n")); |
| 136 | } |  |
| 137 |  |  |
| 138 | vo | id initlog ( const \_ TCHAR \* prog ) { |
| 139 |  | \_ TCHAR logname [ 255 ]; |
| 140 |  | wcscpy\_ s ( logname , prog ); |
| 141 |  |  |
| 142 |  | *// replace extension* |
| 143 |  | \_ TCHAR \* extension ; |
| 144 |  | extension = wcsstr ( logname , \_T(". exe ")); |
| 145  146 |  | wcsncpy\_ s ( extension , 5 , \_T(". log "), 4); |
| 147 |  | *// Try to open log file for append* |
| 148 |  | if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) { |
| 149  150 |  | \_ wperror (\_T(" The following error occurred "));  \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname ); |
| 151 |  | exit (1) ; |
| 152 |  | } |
| 153 |  |  |

154 writelog (\_T("% s is starting ."), prog );

### 155 }

### 156

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

### 160 }

### 161

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;

### 165

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 168

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

### 173

1. *// Convert to normal representation .*
2. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
3. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
4. newtime . tm\_min , newtime . tm\_ sec );

### 178

1. *// Write date and time*
2. fwprintf ( logfile , \_T("% s"), buf );
3. *// Write all params*
4. va\_ start ( ap , format );
5. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);
6. fwprintf ( logfile , \_T("% s"), buf );
7. va\_ end ( ap);
8. *// New sting*
9. fwprintf ( logfile , \_T("\ n"));

### 188 }

### 189

1. void syslog ( WORD category , WORD identifier , LPWSTR message ) {
2. LPWSTR p Messages [1] = { message };

### 192

1. if (! Report Event (
2. eventlog , *// event log handle*
3. EVENTLOG\_ INFORMATION\_ TYPE , *// event type*
4. category , *// event category*
5. identifier , *// event identifier*
6. NULL , *// user security identifier*
7. 1 , *// number of substitution strings*
8. 0 , *// data size*
9. ( LPCWSTR \*) pMessages , *// pointer to strings*
10. NULL )) { *// pointer to binary data buffer*
11. writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

### 204 }

### 205 }

## Таким образом, рассмотрены два способа фильтрации исключений - на уровне входа в блок except, либо уже непосредственно в обработчике (тогда в except ставится макро-константа EXCEPTION\_EXECUTE\_HANDLER, позволяющая принимать любые исключения). Далее будет рассмотрен более логичный способ фильтрации исключений специальной функцией.

## Запуск под отладчиком показывает картину практически аналогичную предыдущему случаю, но есть разница в логе работы программы (листинг 6). На этот раз мы знаем какое исключение произошло и это фиксируем в логе.

## Листинг 6: Генерация и обработка исключения с помощью функций WinAPI

1 [ 6 / 2 / 2015 16 : 42 : 36 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Exceptions Processing \ Debug \ Get Exception Code . exe is starting .

2

3

4

5

6

7

[ 6 / 2 / 2015

[ 6 / 2 / 2015 [ 6 / 2 / 2015 [ 6 / 2 / 2015

16 : 42 : 36 ] Task : DIVIDE\_ BY\_ ZERO exception .

16 : 42 : 36 ] Ready for generate DIVIDE\_ BY\_ ZERO exception . 16 : 42 : 36 ] Caught exception is: EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO 16 : 42 : 36 ] Shutting down .

[ 6 / 2 / 2015

16 : 42 : 39 ]

C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Exceptions Processing \ Debug \ Get Exception Code . exe is starting .

1. [ 6 / 2 / 2015 16 : 42 : 39 ] Task : FLT\_ OVERFLOW exception .
2. [ 6 / 2 / 2015 16 : 42 : 39 ] Ready for generate FLT\_ OVERFLOW exception .
3. [ 6 / 2 / 2015 16 : 42 : 39 ] Caught exception is: EXCEPTION\_ FLT\_ OVERFLOW

11 [ 6 / 2 / 2015 16 : 42 : 39 ] Shutting down .

# Пользовательская функция-фильтр

## В листинге 7 представлена функция-фильтр, которая возвращает EXCEPTION\_CONTINUE\_SEARCH только если исключение вызвано EXCEPTION\_FLT\_DIVIDE\_BY\_ZERO или EXCEPTION\_FLT\_OVERFLOW.

## Листинг 7: Использование собственной функции фильтра (src/ExceptionsProcessing/FilterFunction/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

*/\**

*Task 3.*

*Create your own filter function .*

*\*/*

*//*

*//*

*//*

*IMPORTANT :*

*Properties*

*Don ’ t forget to disable Enhanced Instructions !!!*

*-> Configuration*

*Properties -> C/ C++ -> Code Generation ->*

*Enable Enhanced Instruction*

*Set*

*=*

*No*

*Enhanced*

*Instructions*

*(/ arch: IA32 )*

# include

# include

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cstring >

< cfloat >

< cmath >

< excpt . h>

< windows . h>

< time . h>

# include

" messages . h"

*// log*

FILE \* logfile ; HANDLE eventlog ;

void void void void LONG

usage ( const \_ TCHAR \* prog );

initlog ( const \_ TCHAR \* prog ); closelog ();

writelog ( \_ TCHAR \* format , ...) ;

Filter ( DWORD dw Exception Gode );

29 void syslog ( WORD category , WORD identifier , LPWSTR message );

### 30

1. *// Task switcher*
2. enum {
3. DIVIDE\_ BY\_ ZERO ,
4. FLT\_ OVERFLOW
5. } task ;

### 36

1. *// Defines the entry point for the console application .*
2. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
3. *// Init log*
4. initlog ( argv [0]) ;
5. eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 42

1. *// Check parameters number*
2. if ( argc != 2) {
3. \_ tprintf (\_T(" Too few parameters .\ n\ n"));
4. writelog (\_T(" Too few parameters ."));
5. usage ( argv [0]) ;
6. closelog ();
7. exit (1) ;

50 }

### 51

52 *// Set task*

53 if (! \_ tcscmp (\_T("- d"), argv [1]) ) {

1. task = DIVIDE\_ BY\_ ZERO ;
2. writelog (\_T(" Task : DIVIDE\_ BY\_ ZERO exception ."));

56 }

1. else if (! \_ tcscmp (\_T("-o"), argv [1]) ) {
2. task = FLT\_ OVERFLOW ;
3. writelog (\_T(" Task : FLT\_ OVERFLOW exception ."));

60 }

1. else {
2. \_ tprintf (\_T(" Can ’t parse parameters .\ n\ n"));
3. writelog (\_T(" Can ’t parse parameters ."));
4. usage ( argv [0]) ;
5. closelog ();
6. exit (1) ;

67 }

### 68

1. *// Floating point exceptions are masked by default .*
2. \_ clearfp ();
3. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 72

73 *// Set exception*

1. \_\_ try {
2. volatile float tmp = 0;
3. switch ( task ) {
4. case DIVIDE\_ BY\_ ZERO :
5. writelog (\_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
6. syslog ( ZERODIVIDE\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
7. \_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
8. tmp = 1 / tmp ;
9. writelog (\_T(" DIVIDE\_ BY\_ ZERO exception is generated ."));
10. break ;
11. case FLT\_ OVERFLOW :
12. *// Note: floating point execution happens asynchronously .*
13. *// So , the exception will not be handled until the next floating*
14. *// point instruction .*
15. writelog (\_T(" Ready for generate FLT\_ OVERFLOW exception ."));
16. syslog ( OVERFLOW\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
17. \_T(" Ready for generate FLT\_ OVERFLOW exception ."));
18. tmp = pow ( FLT\_MAX , 3);
19. writelog (\_T(" Task : FLT\_ OVERFLOW exception is generated ."));
20. break ;
21. default :
22. break ;

96 }

97 }

1. *// Own filter function .*
2. \_\_ except ( Filter ( Get Exception Code ())) {
3. printf (" Caught exception is: ");
4. switch ( Get Exception Code ()){
5. case EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO :
6. \_ tprintf (\_T(" Caught exception is: EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO "));
7. writelog (\_T(" Caught exception is: EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO "));
8. syslog ( ZERODIVIDE\_ CATEGORY , CAUGHT\_ EXCEPRION ,

|  |  |  |
| --- | --- | --- |
| 106 |  | \_T(" Caught exception is: EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO .")); |
| 107 |  | break ; |
| 108 |  | case EXCEPTION\_ FLT\_ OVERFLOW : |
| 109 |  | \_ tprintf (\_T(" Caught exception is: EXCEPTION\_ FLT\_ OVERFLOW ")); |
| 110 |  | writelog (\_T(" Caught exception is: EXCEPTION\_ FLT\_ OVERFLOW ")); |
| 111 |  | syslog ( OVERFLOW\_ CATEGORY , CAUGHT\_ EXCEPRION , |
| 112 |  | \_T(" Caught exception is: EXCEPTION\_ FLT\_ OVERFLOW .")); |
| 113 |  | break ; |
| 114 |  | default : |
| 115 |  | \_ tprintf (\_T(" UNKNOWN exception : % x\ n"), Get Exception Code ()); |
| 116 |  | writelog (\_T(" UNKNOWN exception : % x\ n"), Get Exception Code ()); |
| 117 |  | } |
| 118 | } |  |

1. closelog ();
2. Close Handle ( eventlog );
3. exit (0) ;

### 122 }

### 123

1. *// Own filter function .*
2. LONG Filter ( DWORD dw Exception Gode ) {
3. \_ tprintf (\_T(" Filter function used "));
4. if ( dw Exception Gode == EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO || dw Exception Gode == EXCEPTION\_ FLT\_ OVERFLOW )
5. return EXCEPTION\_ EXECUTE\_ HANDLER ;
6. return EXCEPTION\_ CONTINUE\_ SEARCH ;

### 130 }

### 131

1. *// Usage manual*
2. void usage ( const \_ TCHAR \* prog ) {
3. \_ tprintf (\_T(" Usage : \ n"));
4. \_ tprintf (\_T("\ t% s -d\ n"), prog );
5. \_ tprintf (\_T("\ t\ t\ t for exception float divide by zero ,\ n"));
6. \_ tprintf (\_T("\ t% s -o\ n"), prog );
7. \_ tprintf (\_T("\ t\ t\ t for exception float overflow .\ n"));

### 139 }

### 140

1. void initlog ( const \_ TCHAR \* prog ) {
2. \_ TCHAR logname [ 255 ];
3. wcscpy\_ s ( logname , prog );

### 144

1. *// replace extension*
2. \_ TCHAR \* extension ;
3. extension = wcsstr ( logname , \_T(". exe "));
4. wcsncpy\_ s ( extension , 5 , \_T(". log "), 4);

### 149

1. *// Try to open log file for append*
2. if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {
3. \_ wperror (\_T(" The following error occurred "));
4. \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname );
5. exit (1) ;

### 155 }

### 156

157 writelog (\_T("% s is starting ."), prog );

### 158 }

### 159

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

### 163 }

### 164

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;

### 168

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 171

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

### 176

1. *// Convert to normal representation .*
2. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
3. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
4. newtime . tm\_min , newtime . tm\_ sec );

### 181

1. *// Write date and time*
2. fwprintf ( logfile , \_T("% s"), buf );
3. *// Write all params*
4. va\_ start ( ap , format );
5. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);
6. fwprintf ( logfile , \_T("% s"), buf );
7. va\_ end ( ap);
8. *// New sting*
9. fwprintf ( logfile , \_T("\ n"));

### 191 }

### 192

1. void syslog ( WORD category , WORD identifier , LPWSTR message ) {
2. LPWSTR p Messages [1] = { message };

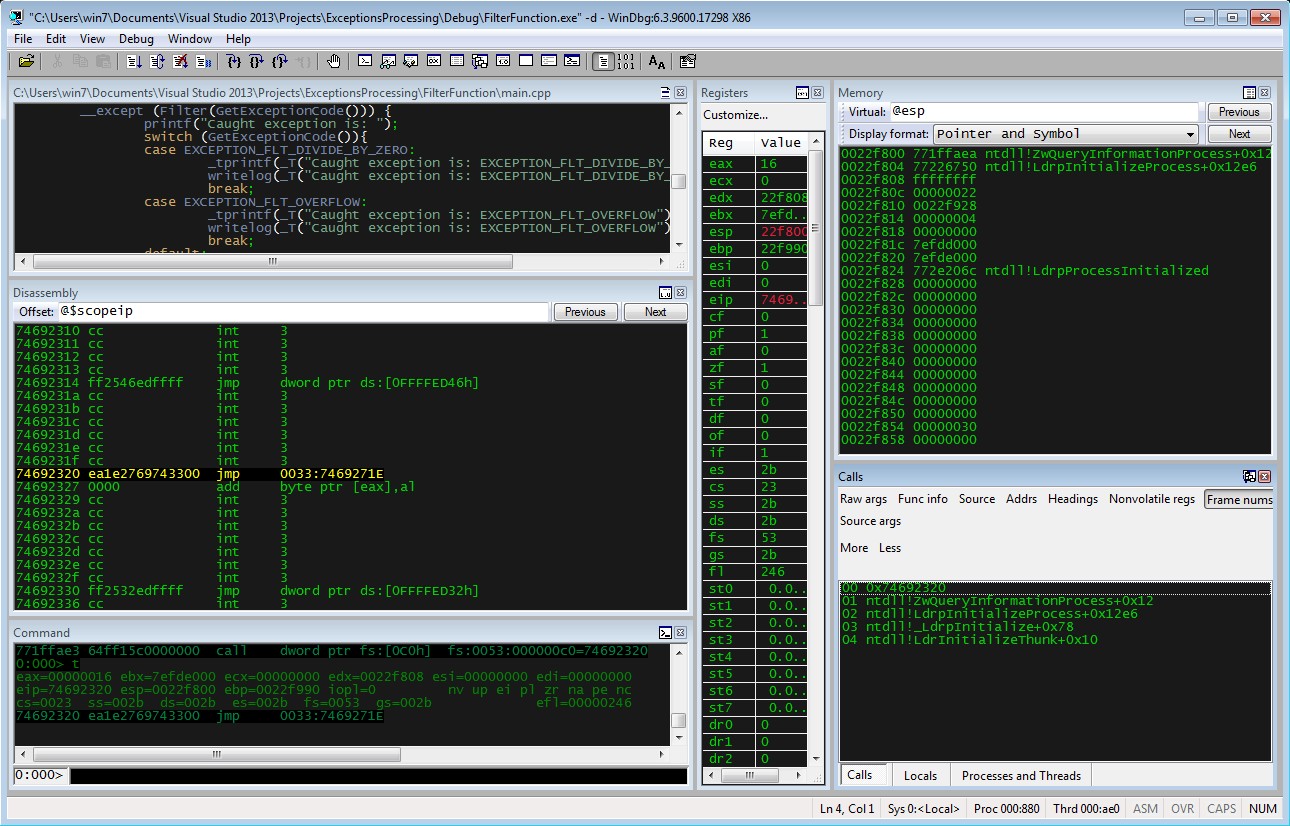
### 195

1. if (! Report Event (
2. eventlog , *// event log handle*
3. EVENTLOG\_ INFORMATION\_ TYPE , *// event type*
4. category , *// event category*
5. identifier , *// event identifier*
6. NULL , *// user security identifier*
7. 1 , *// number of substitution strings*
8. 0 , *// data size*
9. ( LPCWSTR \*) pMessages , *// pointer to strings*
10. NULL )) { *// pointer to binary data buffer*
11. writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

### 207 }

208 }

## При изучении работы программы под отладчиком, выяснилось что при возникновении исключения, управление не передаётся в то место, где определена функция-фильтр. Веро- ятно компилятор оптимизирует код и подставляет её целиком на место вызова. На рисунке 10 видна передача управления на обработку исключения после работы функции-фильтра.



## Рис. 10: Передача управления обработчику, после отработки функции-фильтра

## В обработчике фактически происходит только вызов функций логирования (листинг 8). Как и раньше, строки 82 и 92 оказываются пропущенными, т.к. после возбуждения исключения управление переходит обработчику, нарушая линейный порядок.

## Листинг 8: Генерация и обработка исключения с помощью функций WinAPI

1 [ 6 / 2 / 2015 16 : 45 : 29 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Exceptions Processing \ Debug \ Filter Function . exe is starting .

2

3

4

5

6

7

[ 6 / 2 / 2015

[ 6 / 2 / 2015 [ 6 / 2 / 2015 [ 6 / 2 / 2015

16 : 45 : 29 ] Task : DIVIDE\_ BY\_ ZERO exception .

16 : 45 : 29 ] Ready for generate DIVIDE\_ BY\_ ZERO exception . 16 : 45 : 29 ] Caught exception is: EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO 16 : 45 : 29 ] Shutting down .

[ 6 / 2 / 2015

16 : 45 : 34 ]

C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Exceptions Processing \ Debug \ Filter Function . exe is starting .

1. [ 6 / 2 / 2015 16 : 45 : 34 ] Task : FLT\_ OVERFLOW exception .
2. [ 6 / 2 / 2015 16 : 45 : 34 ] Ready for generate FLT\_ OVERFLOW exception .
3. [ 6 / 2 / 2015 16 : 45 : 34 ] Caught exception is: EXCEPTION\_ FLT\_ OVERFLOW

11 [ 6 / 2 / 2015 16 : 45 : 34 ] Shutting down .

# Использование RaiseException

## Исключение можно возбудить не только в результате каких-то арифметических или ло- гических операций, но и искусственным образом, вызвав функцию RaiseException. Она обладает 4-я параметрами, но наиболее важным является первый, который определяет тип возбуждаемого исключения. Работа этой функции показана в листинге 9.

## Листинг 9: Программная генерация исключения при помощи функции RaiseException (src/ExceptionsProcessing/RaiseException/main.cpp)

1 */\**

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

*Task 4.*

*Get information about the exception using the Get Exception Information () fnc;*

*throw an exception using the Raise Exception () function .*

*\*/*

*//*

*//*

*//*

*IMPORTANT : Don ’ t forget to disable Enhanced Instructions !!!*

*Properties -> Configuration*

*Enable Enhanced Instruction*

*Properties -> C/ C++ -> Code Generation ->*

*Set*

*=*

*No*

*Enhanced*

*Instructions*

*(/ arch: IA32 )*

# include

# include

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cstring >

< cfloat >

< cmath >

< excpt . h>

< windows . h>

< time . h>

# include

" messages . h"

*// log*

FILE \* logfile ; HANDLE eventlog ;

void usage ( const \_ TCHAR \* prog ); void initlog ( const \_ TCHAR \* prog );

1. void closelog ();
2. void writelog ( \_ TCHAR \* format , ...) ;
3. LONG Filter ( DWORD dw Exception Gode , const \_ EXCEPTION\_ POINTERS \* ep);
4. void syslog ( WORD category , WORD identifier , LPWSTR message );

### 31

1. *// Task switcher*
2. enum {
3. DIVIDE\_ BY\_ ZERO ,
4. FLT\_ OVERFLOW
5. } task ;

### 37

1. *// Defines the entry point for the console application .*
2. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
3. *// Init log*
4. initlog ( argv [0]) ;
5. eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 43

1. *// Check parameters number*
2. if ( argc != 2) {
3. \_ tprintf (\_T(" Too few parameters .\ n\ n"));
4. writelog (\_T(" Too few parameters ."));
5. usage ( argv [0]) ;
6. closelog ();
7. exit (1) ;

51 }

### 52

53 *// Set task*

54 if (! \_ tcscmp (\_T("- d"), argv [1]) ) {

1. task = DIVIDE\_ BY\_ ZERO ;
2. writelog (\_T(" Task : DIVIDE\_ BY\_ ZERO exception ."));

57 }

1. else if (! \_ tcscmp (\_T("-o"), argv [1]) ) {
2. task = FLT\_ OVERFLOW ;
3. writelog (\_T(" Task : FLT\_ OVERFLOW exception ."));

61 }

1. else {
2. \_ tprintf (\_T(" Can ’t parse parameters .\ n\ n"));
3. writelog (\_T(" Can ’t parse parameters ."));
4. usage ( argv [0]) ;
5. closelog ();
6. exit (1) ;

68 }

### 69

1. *// Floating point exceptions are masked by default .*
2. \_ clearfp ();
3. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 73

1. \_\_ try {
2. switch ( task ) {
3. case DIVIDE\_ BY\_ ZERO :
4. *// throw an exception using the Raise Exception () function*
5. writelog (\_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
6. syslog ( ZERODIVIDE\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
7. \_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
8. Raise Exception ( EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ,
9. EXCEPTION\_ NONCONTINUABLE , 0 , NULL );
10. writelog (\_T(" DIVIDE\_ BY\_ ZERO exception is generated ."));
11. break ;
12. case FLT\_ OVERFLOW :
13. *// throw an exception using the Raise Exception () function*
14. writelog (\_T(" Ready for generate FLT\_ OVERFLOW exception ."));
15. syslog ( OVERFLOW\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
16. \_T(" Ready for generate FLT\_ OVERFLOW exception ."));
17. Raise Exception ( EXCEPTION\_ FLT\_ OVERFLOW ,
18. EXCEPTION\_ NONCONTINUABLE , 0 , NULL );
19. writelog (\_T(" Task : FLT\_ OVERFLOW exception is generated ."));
20. break ;
21. default :
22. break ;

96 }

97 }

1. \_\_ except ( Filter ( Get Exception Code () , Get Exception Information ())) {
2. *// There is nothing to do , everything is done in the filter function .*

|  |  |  |
| --- | --- | --- |
| 100 |  | } |
| 101 |  | closelog (); |
| 102 |  | Close Handle ( eventlog ); |
| 103 |  | exit (0) ; |
| 104 | } |  |
| 105 |  |  |
| 106 | LO | NG Filter ( DWORD dw Exception Gode , const \_ EXCEPTION\_ POINTERS \* |

Exception Pointers ) {

1. enum { size = 200 };
2. \_ TCHAR buf [ size ] = { ’\0 ’ };
3. const \_ TCHAR \* err = \_T(" Fatal error !\ nexeption code : 0 x");
4. const \_ TCHAR \* mes = \_T("\ n Program terminate !");
5. if ( Exception Pointers )
6. *// Get information about the exception using the Get Exception Information*
7. swprintf\_ s ( buf , \_T("% s% x% s% x% s% x"), err ,
8. Exception Pointers -> Exception Record -> Exception Code ,
9. \_T(", data adress : 0 x"),

|  |  |  |
| --- | --- | --- |
| 116 |  | Exception Pointers -> Exception Record -> Exception Information [1] , |
| 117 |  | \_T(", instruction adress : 0 x"), |
| 118 |  | Exception Pointers -> Exception Record -> Exception Address , mes ); |
| 119 |  | else |
| 120 |  | swprintf\_ s ( buf , \_T("% s% x% s"), err , dw Exception Gode , mes ); |
| 121 |  |  |
| 122 |  | \_ tprintf (\_T("% s"), buf ); |
| 123 |  | writelog (\_T("% s"), buf ); |
| 124 |  | syslog ( OVERFLOW\_ CATEGORY , CAUGHT\_ EXCEPRION , buf ); |
| 125 |  |  |
| 126 |  | return EXCEPTION\_ EXECUTE\_ HANDLER ; |
| 127 | } |  |
| 128 |  |  |
| 129 | *//* | *Usage manual* |
| 130 | vo | id usage ( const \_ TCHAR \* prog ) { |
| 131 |  | \_ tprintf (\_T(" Usage : \ n")); |
| 132 |  | \_ tprintf (\_T("\ t% s -d\ n"), prog ); |
| 133 |  | \_ tprintf (\_T("\ t\ t\ t for exception float divide by zero ,\ n")); |
| 134 |  | \_ tprintf (\_T("\ t% s -o\ n"), prog ); |
| 135 |  | \_ tprintf (\_T("\ t\ t\ t for exception float overflow .\ n")); |
| 136 | } |  |
| 137 |  |  |
| 138 | vo | id initlog ( const \_ TCHAR \* prog ) { |
| 139 |  | \_ TCHAR logname [ 255 ]; |
| 140 |  | wcscpy\_ s ( logname , prog ); |
| 141 |  |  |
| 142 |  | *// replace extension* |
| 143 |  | \_ TCHAR \* extension ; |
| 144 |  | extension = wcsstr ( logname , \_T(". exe ")); |
| 145  146 |  | wcsncpy\_ s ( extension , 5 , \_T(". log "), 4); |
| 147 |  | *// Try to open log file for append* |
| 148 |  | if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) { |
| 149  150 |  | \_ wperror (\_T(" The following error occurred "));  \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname ); |
| 151 |  | exit (1) ; |
| 152 |  | } |
| 153 |  |  |

154 writelog (\_T("% s is starting ."), prog );

### 155 }

### 156

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

### 160 }

### 161

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;

### 165

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 168

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

### 173

1. *// Convert to normal representation .*
2. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
3. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
4. newtime . tm\_min , newtime . tm\_ sec );

### 178

1. *// Write date and time*
2. fwprintf ( logfile , \_T("% s"), buf );
3. *// Write all params*
4. va\_ start ( ap , format );
5. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);
6. fwprintf ( logfile , \_T("% s"), buf );
7. va\_ end ( ap);
8. *// New sting*
9. fwprintf ( logfile , \_T("\ n"));

### 188 }

### 189

1. void syslog ( WORD category , WORD identifier , LPWSTR message ) {
2. LPWSTR p Messages [1] = { message };

### 192

1. if (! Report Event (
2. eventlog , *// event log handle*
3. EVENTLOG\_ INFORMATION\_ TYPE , *// event type*
4. category , *// event category*
5. identifier , *// event identifier*
6. NULL , *// user security identifier*
7. 1 , *// number of substitution strings*
8. 0 , *// data size*
9. ( LPCWSTR \*) pMessages , *// pointer to strings*
10. NULL )) { *// pointer to binary data buffer*
11. writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

### 204 }

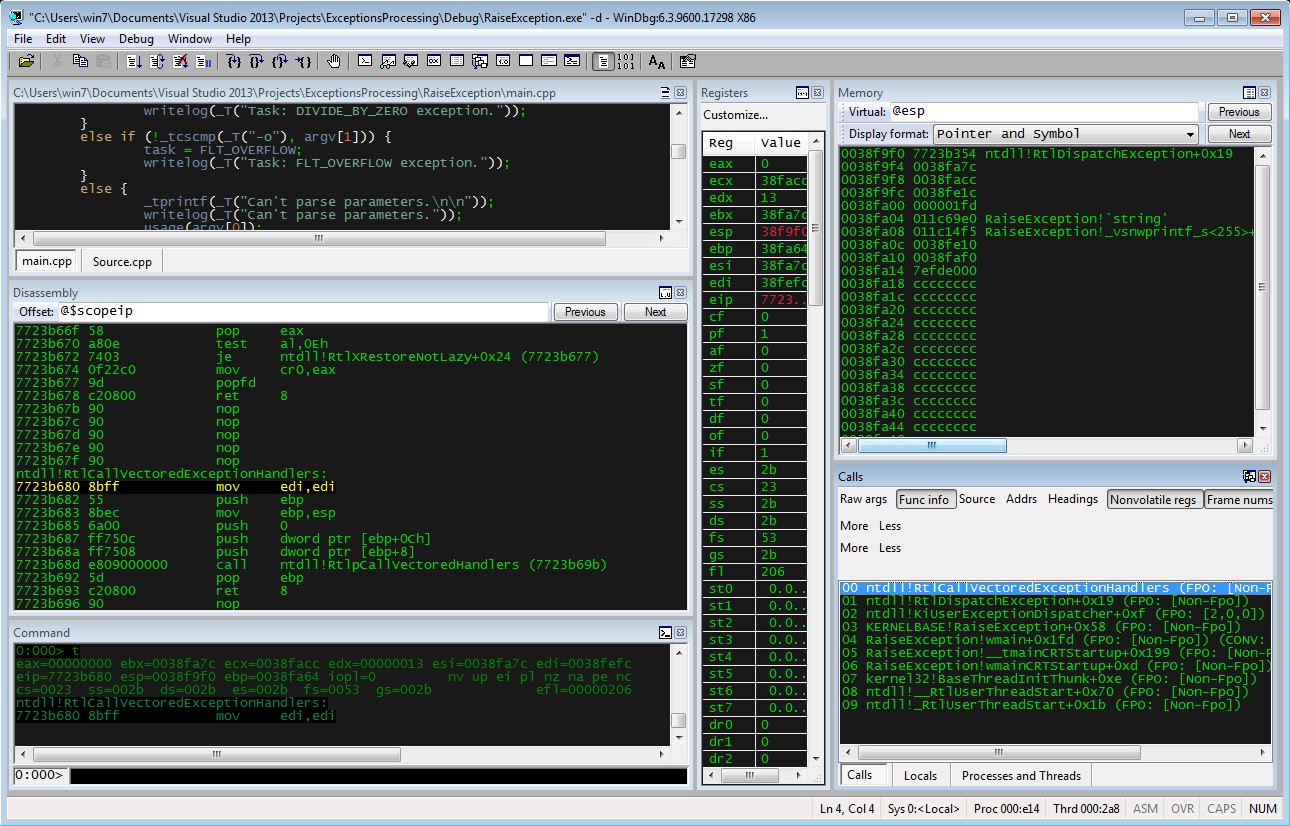
### 205 }

## Информацию об исключении можно получить из функции GetExceptionInformation, ко- торая, в действительности, никакой информацией не владеет но возвращает указатель на структуру EXCEPTION\_POINTERS. В свою очередь, эта структура содержит два указателя на ExceptionRecord и на ContextRecord, в которых уже находится информация об исключении.

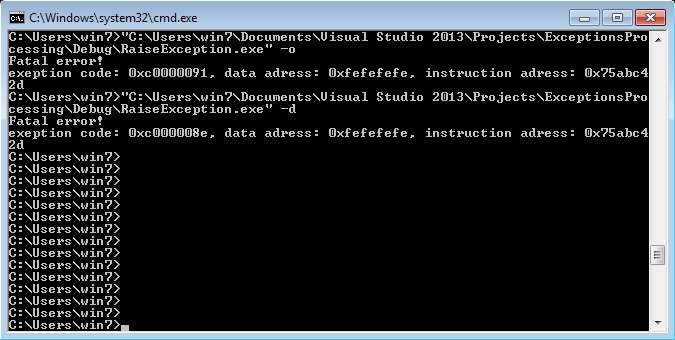
## Важной особенностью функции GetExceptionlnformation является то, что ее можно вызы-

## вать только в функции-фильтре исключений, т.к. структуры CONTEXT, EXCEPTION\_RECORD и EXCEPTION\_POINTERS существуют лишь во время обработки фильтра исключения.

## В момент, когда управление переходит к обработчику исключений, эти данные в стеке разрушаются. На рисунке 11 показан момент получения информации о возникшем исклю- чении. Обработчик исключения находится выше по стеку, и когда ему будет возвращено управление от функции фильтра стек уже будет зачищен.



## Рис. 11: Информация об исключении доступна в процессе работы функции-фильтра



## Рис. 12: Передача управления обработчику, после отработки функции-фильтра

## На рисунке 12 показан вызов программы, генерирующей исключения программным образом, а в листинге 10 представлен лог её работы.

## Листинг 10: Программная генерация исключения и получение информации о нём

1 [ 6 / 2 / 2015 17 : 7 : 22 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Exceptions Processing \ Debug \ Raise Exception . exe is starting .

2

3

4

5

[ 6 / 2 / 2015

[ 6 / 2 / 2015 [ 6 / 2 / 2015

17 : 7 : 22 ] Task : FLT\_ OVERFLOW exception .

17 : 7 : 22 ] Ready for generate FLT\_ OVERFLOW exception . 17 : 7 : 22 ] Fatal error !

6

7

8

exeption code : 0 xc0000091 , data adress : 0 xfefefefe , instruction adress : 0

x 75 abc 42 d

[ 6 / 2 / 2015 17 : 7 : 22 ] Shutting down .

[ 6 / 2 / 2015 17 : 7 : 26 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Exceptions Processing \ Debug \ Raise Exception . exe is starting .

9

10

11

12

[ 6 / 2 / 2015

[ 6 / 2 / 2015 [ 6 / 2 / 2015

17 : 7 : 26 ]

17 : 7 : 26 ]

17 : 7 : 26 ]

Task :

Ready Fatal

DIVIDE\_ BY\_ ZERO exception .

for generate DIVIDE\_ BY\_ ZERO exception . error !

13

exeption code : 0 xc000008e , data adress : 0 xfefefefe , instruction adress : 0

x 75 abc 42 d

[ 6 / 2 / 2015 17 : 7 : 26 ] Shutting down .

# Необрабатываемые исключения

## Если ни один из установленных программистом обработчиков не подошла для обработ- ки исключения (либо программист вообще не установил ни один обработчик), то вы- зывается функция UnhandledExceptionFilter, которая выполняет проверку, запущен ли процесс под отладчиком, и информирует процесс, если отладчик доступен. Далее, функ- ция вызывает фильтр умалчиваемого обработчика (который устанавливается функцией SetUnhandledExceptionFilter и который возвращает EXCEPTION\_EXECUTE\_HANDLER). Затем, в зависимости от настроек операционной системы, вызывается либо отладчик, либо функция NtRaiseHardError, которая отображает сообщение об ошибке.

## Листинг 11 показывает работу с UnhandledExceptionFilter. Возвращаемое значение опреде- ляется в строках 119 и 120.

## Листинг 11: Необработанные исключения (src/ExceptionsProcessing/UnhandleExceptionFilter/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

*/\**

*Task 5.*

*Use the Unhandled Exception Filter and Set Unhandledexceptionfilter for unhandled exceptions ;.*

*\*/*

*//*

*//*

*//*

*IMPORTANT : Don ’ t forget to disable Enhanced Instructions !!!*

*Properties -> Configuration Properties -> C/ C++ -> Code Generation -> Enable Enhanced Instruction Set = No Enhanced Instructions (/ arch: IA32 )*

# include

# include

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cstring >

< cfloat >

< cmath >

< excpt . h>

< windows . h>

< time . h>

# include

" messages . h"

### 20

1. *// log*
2. FILE \* logfile ;
3. HANDLE eventlog ;

### 24

1. void usage ( const \_ TCHAR \* prog );
2. void initlog ( const \_ TCHAR \* prog );
3. void closelog ();
4. void writelog ( \_ TCHAR \* format , ...) ;
5. void syslog ( WORD category , WORD identifier , LPWSTR message );

### 30

1. *// protype*
2. LONG WINAPI My Unhandled Exception Filter ( EXCEPTION\_ POINTERS \* Exception Info );

### 33

1. *// Task switcher*
2. enum {
3. DIVIDE\_ BY\_ ZERO ,
4. FLT\_ OVERFLOW
5. } task ;

### 39

1. *// Defines the entry point for the console application .*
2. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
3. *// Init log*
4. initlog ( argv [0]) ;
5. eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 45

1. *// Check parameters number*
2. if ( argc != 2) {
3. \_ tprintf (\_T(" Too few parameters .\ n\ n"));
4. writelog (\_T(" Too few parameters ."));
5. usage ( argv [0]) ;
6. closelog ();
7. exit (1) ;

53 }

### 54

55 *// Set task*

56 if (! \_ tcscmp (\_T("- d"), argv [1]) ) {

1. task = DIVIDE\_ BY\_ ZERO ;
2. writelog (\_T(" Task : DIVIDE\_ BY\_ ZERO exception ."));

59 }

1. else if (! \_ tcscmp (\_T("-o"), argv [1]) ) {
2. task = FLT\_ OVERFLOW ;
3. writelog (\_T(" Task : FLT\_ OVERFLOW exception ."));

63 }

64 else {

1. \_ tprintf (\_T(" Can ’t parse parameters .\ n\ n"));
2. writelog (\_T(" Can ’t parse parameters ."));
3. usage ( argv [0]) ;
4. closelog ();
5. exit (1) ;

70 }

### 71

1. *// Floating point exceptions are masked by default .*
2. \_ clearfp ();
3. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 75

1. volatile float tmp = 0;
2. :: Set Unhandled Exception Filter ( My Unhandled Exception Filter );

### 78

1. switch ( task ) {
2. case DIVIDE\_ BY\_ ZERO :
3. *// throw an exception using the Raise Exception () function*
4. writelog (\_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
5. syslog ( ZERODIVIDE\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
6. \_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
7. Raise Exception ( EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ,
8. EXCEPTION\_ EXECUTE\_ FAULT , 0 , NULL );
9. writelog (\_T(" DIVIDE\_ BY\_ ZERO exception is generated ."));
10. break ;
11. case FLT\_ OVERFLOW :
12. *// throw an exception using the Raise Exception () function*
13. writelog (\_T(" Ready for generate FLT\_ OVERFLOW exception ."));
14. syslog ( OVERFLOW\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
15. \_T(" Ready for generate FLT\_ OVERFLOW exception ."));
16. Raise Exception ( EXCEPTION\_ FLT\_ OVERFLOW ,
17. EXCEPTION\_ EXECUTE\_ FAULT , 0 , NULL );
18. writelog (\_T(" Task : FLT\_ OVERFLOW exception is generated ."));
19. break ;
20. default :
21. break ;

### 100 }

### 101

1. closelog ();
2. Close Handle ( eventlog );
3. exit (0) ;

### 105 }

### 106

1. LONG WINAPI My Unhandled Exception Filter ( EXCEPTION\_ POINTERS \* Exception Info ) {
2. enum { size = 200 };
3. \_ TCHAR buf [ size ] = { ’\0 ’ };
4. const \_ TCHAR \* err = \_T(" Unhandled exception !\ nexeption code : 0 x");
5. *// Get information about the exception using the Get Exception Information*
6. swprintf\_ s ( buf , \_T("% s% x% s% x% s% x"), err , Exception Info -> Exception Record -> Exception Code ,
7. \_T(", data adress : 0 x"), Exception Info -> Exception Record -> Exception Information [1] ,
8. \_T(", instruction adress : 0 x"), Exception Info -> Exception Record -> Exception Address );
9. \_ tprintf (\_T("% s"), buf );
10. writelog (\_T("% s"), buf );
11. syslog ( OVERFLOW\_ CATEGORY , CAUGHT\_ EXCEPRION , buf );

### 118

1. return EXCEPTION\_ CONTINUE\_ SEARCH ;
2. *// return EXCEPTION\_ EXECUTE\_ HANDLER ;*

### 121 }

### 122

1. *// Usage manual*
2. void usage ( const \_ TCHAR \* prog ) {
3. \_ tprintf (\_T(" Usage : \ n"));
4. \_ tprintf (\_T("\ t% s -d\ n"), prog );
5. \_ tprintf (\_T("\ t\ t\ t for exception float divide by zero ,\ n"));
6. \_ tprintf (\_T("\ t% s -o\ n"), prog );
7. \_ tprintf (\_T("\ t\ t\ t for exception float overflow .\ n"));

### 130 }

### 131

1. void initlog ( const \_ TCHAR \* prog ) {
2. \_ TCHAR logname [ 255 ];
3. wcscpy\_ s ( logname , prog );

### 135

1. *// replace extension*
2. \_ TCHAR \* extension ;
3. extension = wcsstr ( logname , \_T(". exe "));
4. wcsncpy\_ s ( extension , 5 , \_T(". log "), 4);

### 140

1. *// Try to open log file for append*
2. if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {
3. \_ wperror (\_T(" The following error occurred "));
4. \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname );
5. exit (1) ;

### 146 }

### 147

148 writelog (\_T("% s is starting ."), prog );

### 149 }

### 150

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

### 154 }

### 155

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;

### 159

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 162

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

### 167

1. *// Convert to normal representation .*
2. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
3. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
4. newtime . tm\_min , newtime . tm\_ sec );

### 172

1. *// Write date and time*
2. fwprintf ( logfile , \_T("% s"), buf );
3. *// Write all params*
4. va\_ start ( ap , format );
5. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);
6. fwprintf ( logfile , \_T("% s"), buf );
7. va\_ end ( ap);
8. *// New sting*
9. fwprintf ( logfile , \_T("\ n"));
10. fflush ( logfile );

### 183 }

### 184

1. void syslog ( WORD category , WORD identifier , LPWSTR message ) {
2. LPWSTR p Messages [1] = { message };

### 187

1. if (! Report Event (
2. eventlog , *// event log handle*
3. EVENTLOG\_ INFORMATION\_ TYPE , *// event type*
4. category , *// event category*
5. identifier , *// event identifier*
6. NULL , *// user security identifier*
7. 1 , *// number of substitution strings*
8. 0 , *// data size*
9. ( LPCWSTR \*) pMessages , *// pointer to strings*

197

198

199

200

NULL )) {

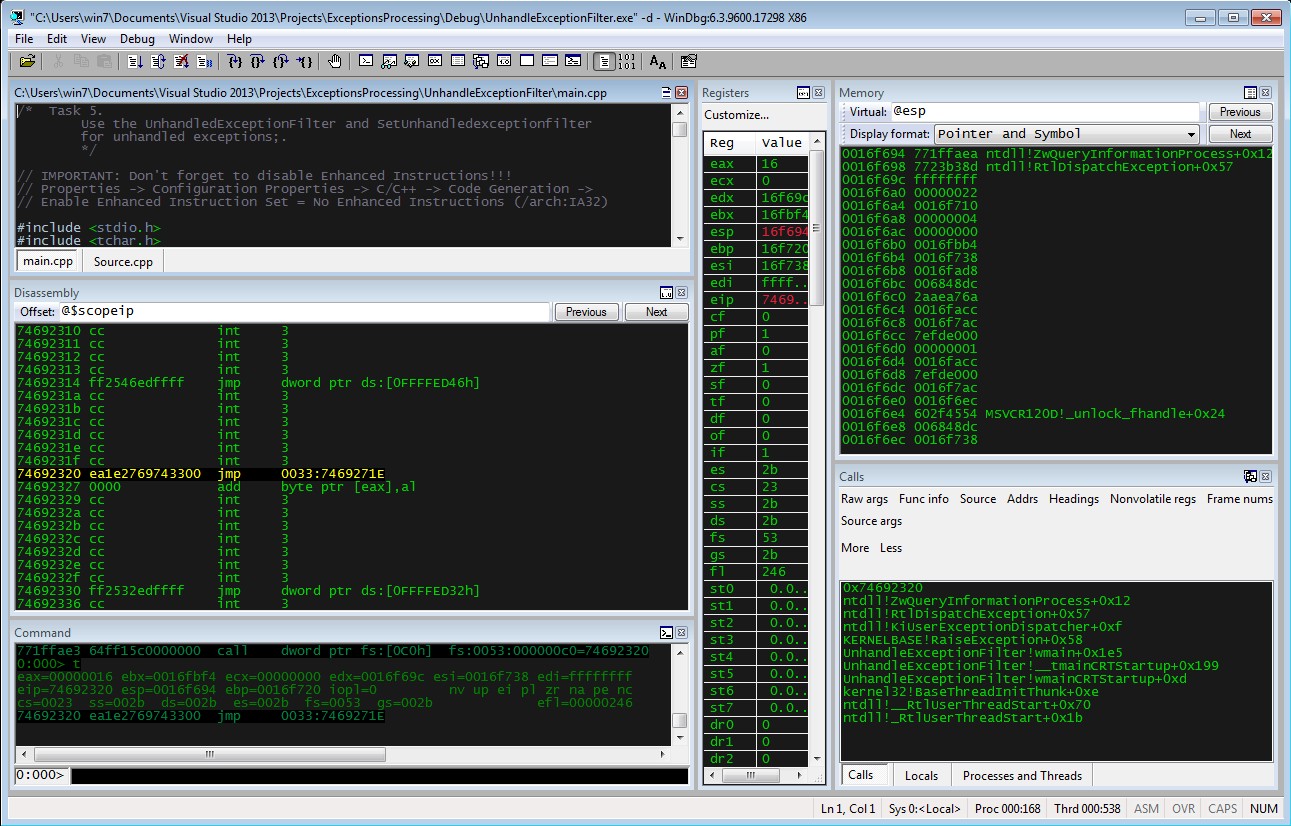
*// pointer to binary data buffer*

writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

}

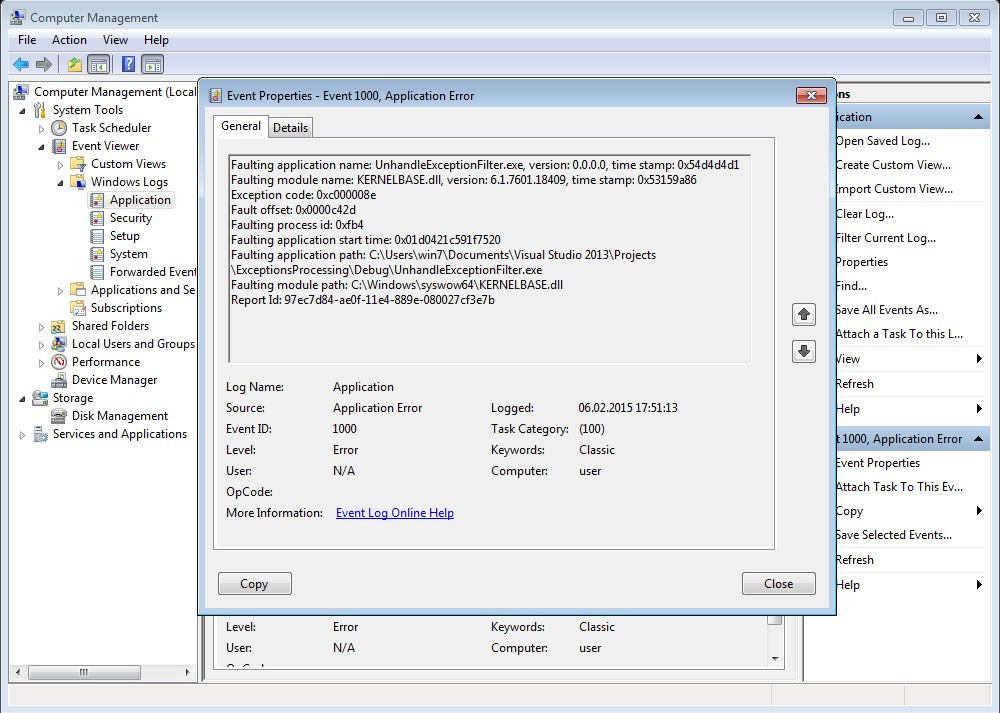
}

## Для начала запустим программу так, чтобы фильтр возвращал EXCEPTION\_EXECUTE\_HANDLE Это считается нормальной ситуацией, и на рисунке 13 видно как происходит передача управления.

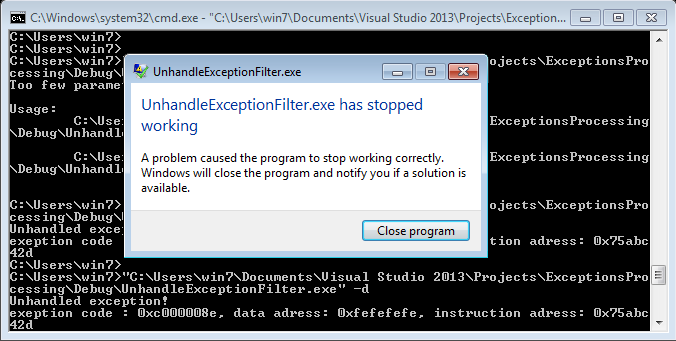


## Рис. 13: Нормальная обработка исключения через фильтр

## Запустим программу ещё раз, фильтр вернёт EXCEPTION\_CONTINUE\_SEARCH. Это событие будет передано операционной системе, и будет зафиксировано в системном журнале (рисунок 14). Что примечательно, обработчик успел выполнить свою задачу, информация об ошибке выведена на экран (рисунок 15) и сохранена в лог (листинг 12), системная ошибка возникла уже после.



## Рис. 14: Исключение зафиксировано в системном журнале



## Рис. 15: Обработчик успел выполниться до системной ошибки

## В логе программы можно прочитать информацию о произошедшей исключительной си- туации. Вместе с тем, можно видеть, что программа не была завершена корректно, а дескриптор файла-лога не был закрыт.

## Листинг 12: Обработчик успел сохранить данные об исключении

1 [ 6 / 2 / 2015 18 : 9 : 43 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \

Exceptions Processing \ Debug \ Unhandle Exception Filter . exe is starting .

2

3

4

5

[ 6 / 2 / 2015

[ 6 / 2 / 2015 [ 6 / 2 / 2015

18 : 9 : 43 ] Task : DIVIDE\_ BY\_ ZERO exception .

18 : 9 : 43 ] Ready for generate DIVIDE\_ BY\_ ZERO exception . 18 : 9 : 43 ] Unhandled exception !

exeption code : 0 xc000008e , data adress : 0 xfefefefe , instruction adress : 0

x 75 abc 42 d

## Из рассмотренного примера становится видно, что возврат EXCEPTION\_EXECUTE\_HANDLER является более предпочитаемым результатом, т.к. исключение нужно обрабатывать там,

## где оно возникло.

# Вложенные исключения

## Листинг 13 показывает, как происходит передача исключения, в поисках подходящего обработчика. Самым ближайшим (по стеку) обработчиком для исключения, вызванного делением на 0, является обработчик из 48-й строки. Но там стоит ограничение, позволяющее обрабатывать только исключения, вызванные переполнением. В результате обработка этого исключения передаётся в 58-ю строку, хотя этот обработчик дальше по стеку.

## Листинг 13: Вложенные исключения (src/ExceptionsProcessing/NestedException/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

*/\**

*Task 6.*

*Nested exception process;*

*\*/*

*//*

*//*

*//*

*IMPORTANT : Don ’ t forget to disable Enhanced Instructions !!!*

*Properties -> Configuration Properties -> C/ C++ -> Code Generation ->*

*Enable Enhanced Instruction Set*

*=*

*No*

*Enhanced*

*Instructions*

*(/ arch: IA32 )*

# include

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cstring >

< cfloat >

< excpt . h>

< windows . h>

< time . h>

# include

" messages . h"

*// log*

FILE \* logfile ; HANDLE eventlog ;

void void void void

initlog ( const \_ TCHAR \* prog );

closelog ();

writelog ( \_ TCHAR \* format , ...) ;

syslog ( WORD category , WORD identifier , LPWSTR message );

1. *// Defines the entry point for the console application .*
2. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
3. *// Init log*
4. initlog ( argv [0]) ;
5. eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 33

1. *// Floating point exceptions are masked by default .*
2. \_ clearfp ();
3. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 37

1. \_\_ try {
2. \_\_ try {
3. writelog (\_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
4. syslog ( ZERODIVIDE\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
5. \_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
6. Raise Exception ( EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ,
7. EXCEPTION\_ NONCONTINUABLE , 0 , NULL );
8. writelog (\_T(" DIVIDE\_ BY\_ ZERO exception is generated ."));

46 }

1. \_\_ except (( Get Exception Code () == EXCEPTION\_ FLT\_ OVERFLOW ) ?
2. EXCEPTION\_ EXECUTE\_ HANDLER :
3. EXCEPTION\_ CONTINUE\_ SEARCH )

50 {

1. writelog (\_T(" Internal handler in action ."));
2. \_ tprintf (\_T(" Internal handler in action ."));
3. syslog ( OVERFLOW\_ CATEGORY , CAUGHT\_ EXCEPRION ,
4. \_T(" Internal handler in action ."));

55 }

56 }

1. \_\_ except (( Get Exception Code () == EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ) ?
2. EXCEPTION\_ EXECUTE\_ HANDLER :
3. EXCEPTION\_ CONTINUE\_ SEARCH )

60 {

1. writelog (\_T(" External handler in action ."));
2. \_ tprintf (\_T(" External handler in action ."));
3. syslog ( ZERODIVIDE\_ CATEGORY , CAUGHT\_ EXCEPRION ,
4. \_T(" Internal handler in action ."));

65 }

### 66

1. closelog ();
2. Close Handle ( eventlog );
3. exit (0) ;

70 }

### 71

72 void initlog ( const \_ TCHAR \* prog ) {

1. \_ TCHAR logname [ 255 ];
2. wcscpy\_ s ( logname , prog );

### 75

1. *// replace extension*
2. \_ TCHAR \* extension ;
3. extension = wcsstr ( logname , \_T(". exe "));
4. wcsncpy\_ s ( extension , 5 , \_T(". log "), 4);

### 80

1. *// Try to open log file for append*
2. if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {
3. \_ wperror (\_T(" The following error occurred "));
4. \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname );
5. exit (1) ;

86 }

### 87

88 writelog (\_T("% s is starting ."), prog );

89 }

### 90

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

94 }

### 95

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;

### 99

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 102

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

### 107

1. *// Convert to normal representation .*
2. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
3. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
4. newtime . tm\_min , newtime . tm\_ sec );

### 112

1. *// Write date and time*
2. fwprintf ( logfile , \_T("% s"), buf );
3. *// Write all params*
4. va\_ start ( ap , format );
5. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);

|  |  |  |
| --- | --- | --- |
| 118 |  | fwprintf ( logfile , \_T("% s"), buf ); |
| 119 |  | va\_ end ( ap); |
| 120 |  | *// New sting* |
| 121 |  | fwprintf ( logfile , \_T("\ n")); |
| 122 | } |  |
| 123 |  |  |
| 124 | vo | id syslog ( WORD category , WORD identifier , LPWSTR message ) { |
| 125 |  | LPWSTR p Messages [1] = { message }; |
| 126 |  |  |
| 127 |  | if (! Report Event ( |
| 128 |  | eventlog , *// event log handle* |

## Запуск отладчика подтвердил ожидаемый результат - поиск подходящего обработчика для исключения происходит снизу вверх. В начале проверяется ближайший обработчик (на рисунке 16 отрабатывает фильтр ближайшего обработчика исключения). Эта проверка вернёт EXCEPTION\_CONTINUE\_SEARCH для продолжения поиска более подходящего обработчика и передачи управления дальше по стеку.

129

130

131

132

133

134

135

136

137

138

139

EVENTLOG\_ INFORMATION\_ TYPE , *// event type*

category ,

identifier , NULL ,

1 ,

0 ,

*//*

*//*

*//*

*// event category*

*// event identifier user security identifier*

*number of substitution strings*

*data size*

( LPCWSTR \*) pMessages ,

*// pointer to strings*

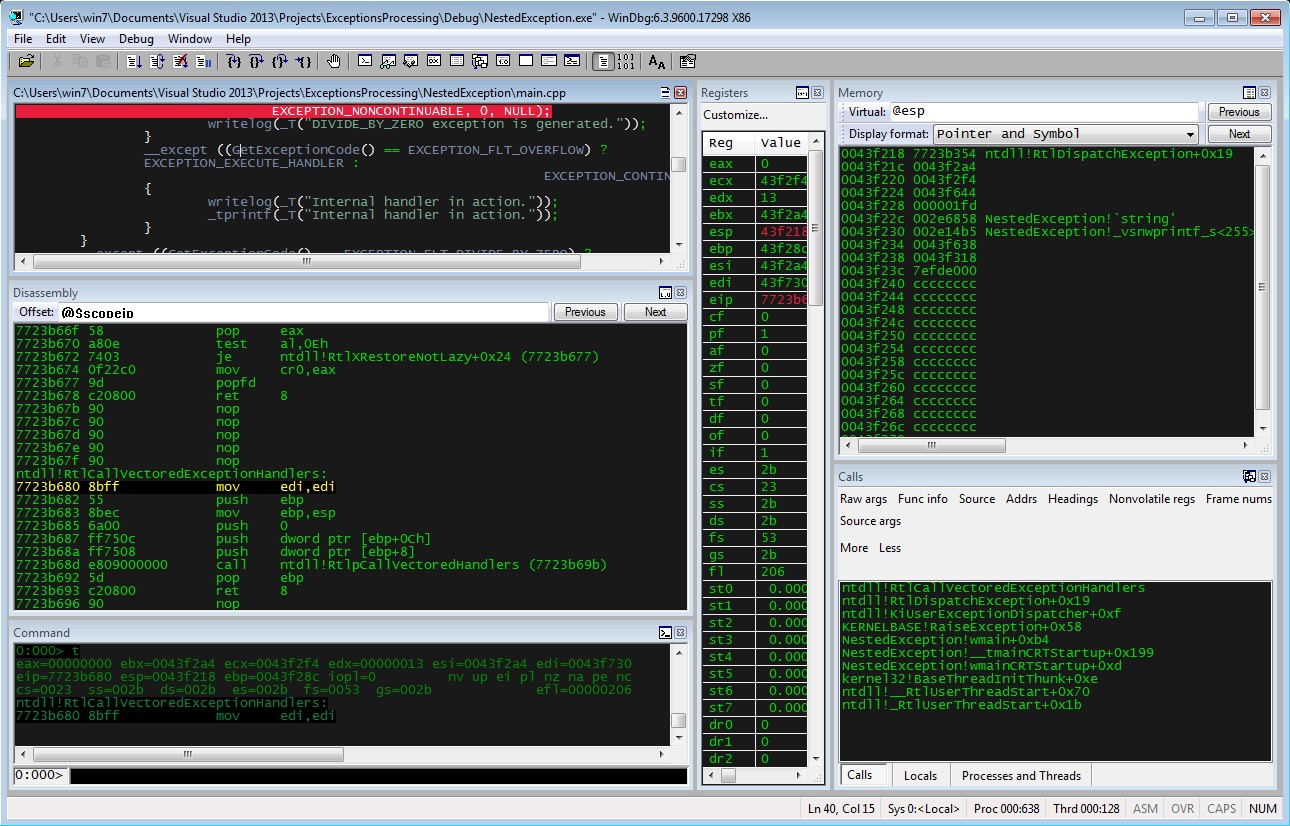
NULL )) {

*// pointer to binary data buffer*

writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

}

}



## Рис. 16: Проверка условий обработчика исключения

## При этом создаётся опасность утечки ресурсов, поэтому желательно обрабатывать исклю- чительные ситуации в месте их возникновения. Протокол работы программы показан в листинге 14.

## Листинг 14: Обработчик успел сохранить данные об исключении

1. [ 6 / 2 / 2015 18 : 43 : 54 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \ Exceptions Processing \ Debug \ Nested Exception . exe is starting .
2. [ 6 / 2 / 2015 18 : 43 : 54 ] Ready for generate DIVIDE\_ BY\_ ZERO exception .
3. [ 6 / 2 / 2015 18 : 43 : 54 ] External handler in action .

4 [ 6 / 2 / 2015 18 : 43 : 54 ] Shutting down .

# Выход при помощи goto

## Использование goto считается дурной практикой по целому ряду причин. В листинге 15, благодаря goto управление со строки 40 передаётся сразу на строку 55. Таким образом осуществляется выход из блока try без возбуждения и обработки исключения.

## Листинг 15: Выход из блока охраняемого кода при помощи goto (src/ExceptionsProcessing/Goto/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

*/\**

*Task 7.*

*Get out of the \_\_ try block by using the goto;*

*\*/*

*//*

*//*

*//*

*IMPORTANT : Don ’ t forget to disable Enhanced Instructions !!!*

*Properties -> Configuration Properties -> C/ C++ -> Code Generation ->*

*Enable Enhanced Instruction Set*

*=*

*No*

*Enhanced*

*Instructions*

*(/ arch: IA32 )*

# include

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cstring >

< cfloat >

< excpt . h>

< windows . h>

< time . h>

# include

" messages . h"

*// log*

FILE \* logfile ; HANDLE eventlog ;

void void void void

initlog ( const \_ TCHAR \* prog );

closelog ();

writelog ( \_ TCHAR \* format , ...) ;

syslog ( WORD category , WORD identifier , LPWSTR message );

*// Defines the entry point for the console application .*

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// Init log*
3. initlog ( argv [0]) ;
4. eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 33

1. *// Floating point exceptions are masked by default .*
2. \_ clearfp ();
3. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 37

1. \_\_ try {
2. writelog (\_T(" Call goto "));
3. goto OUT\_ POINT ;
4. writelog (\_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
5. syslog ( ZERODIVIDE\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
6. \_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
7. Raise Exception ( EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ,
8. EXCEPTION\_ NONCONTINUABLE , 0 , NULL );
9. writelog (\_T(" DIVIDE\_ BY\_ ZERO exception is generated ."));

47 }

48 \_\_ except ( EXCEPTION\_ EXECUTE\_ HANDLER )

49 {

1. writelog (\_T(" Handler in action ."));
2. \_ tprintf (\_T(" Handler in action ."));
3. syslog ( ZERODIVIDE\_ CATEGORY , CAUGHT\_ EXCEPRION ,
4. \_T(" Handler in action ."));

54 }

1. OUT\_ POINT :
2. writelog (\_T(" A point outside the

\_\_try block ."));

1. \_ tprintf (\_T(" A point outside the

\_\_try block ."));

1. syslog ( ZERODIVIDE\_ CATEGORY , CAUGHT\_ EXCEPRION ,
2. \_T(" A point outside the

### 60

\_\_try block ."));

1. closelog ();
2. Close Handle ( eventlog );
3. exit (0) ;

64 }

### 65

1. void initlog ( const \_ TCHAR \* prog ) {
2. \_ TCHAR logname [ 255 ];
3. wcscpy\_ s ( logname , prog );

### 69

1. *// replace extension*
2. \_ TCHAR \* extension ;
3. extension = wcsstr ( logname , \_T(". exe "));
4. wcsncpy\_ s ( extension , 5 , \_T(". log "), 4);

### 74

* 1. *// Try to open log file for append*
  2. if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {
  3. \_ wperror (\_T(" The following error occurred "));
  4. \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname );
  5. exit (1) ;

80 }

### 81

82 writelog (\_T("% s is starting ."), prog );

83 }

### 84

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

88 }

### 89

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;

### 93

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 96

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

### 101

1. *// Convert to normal representation .*
2. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
3. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
4. newtime . tm\_min , newtime . tm\_ sec );

### 106

1. *// Write date and time*
2. fwprintf ( logfile , \_T("% s"), buf );
3. *// Write all params*
4. va\_ start ( ap , format );
5. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);
6. fwprintf ( logfile , \_T("% s"), buf );
7. va\_ end ( ap);
8. *// New sting*
9. fwprintf ( logfile , \_T("\ n"));

### 116 }

### 117

118 void syslog ( WORD category , WORD identifier , LPWSTR message ) {

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

LPWSTR p Messages [1] = { message };

if (! Report Event (

eventlog ,

*// event*

EVENTLOG\_ INFORMATION\_ TYPE ,

category , identifier , NULL ,

1 ,

0 ,

*// event*

*log handle*

*// event type category*

*//*

*//*

*//*

*// event identifier*

*user security identifier number of substitution strings data size*

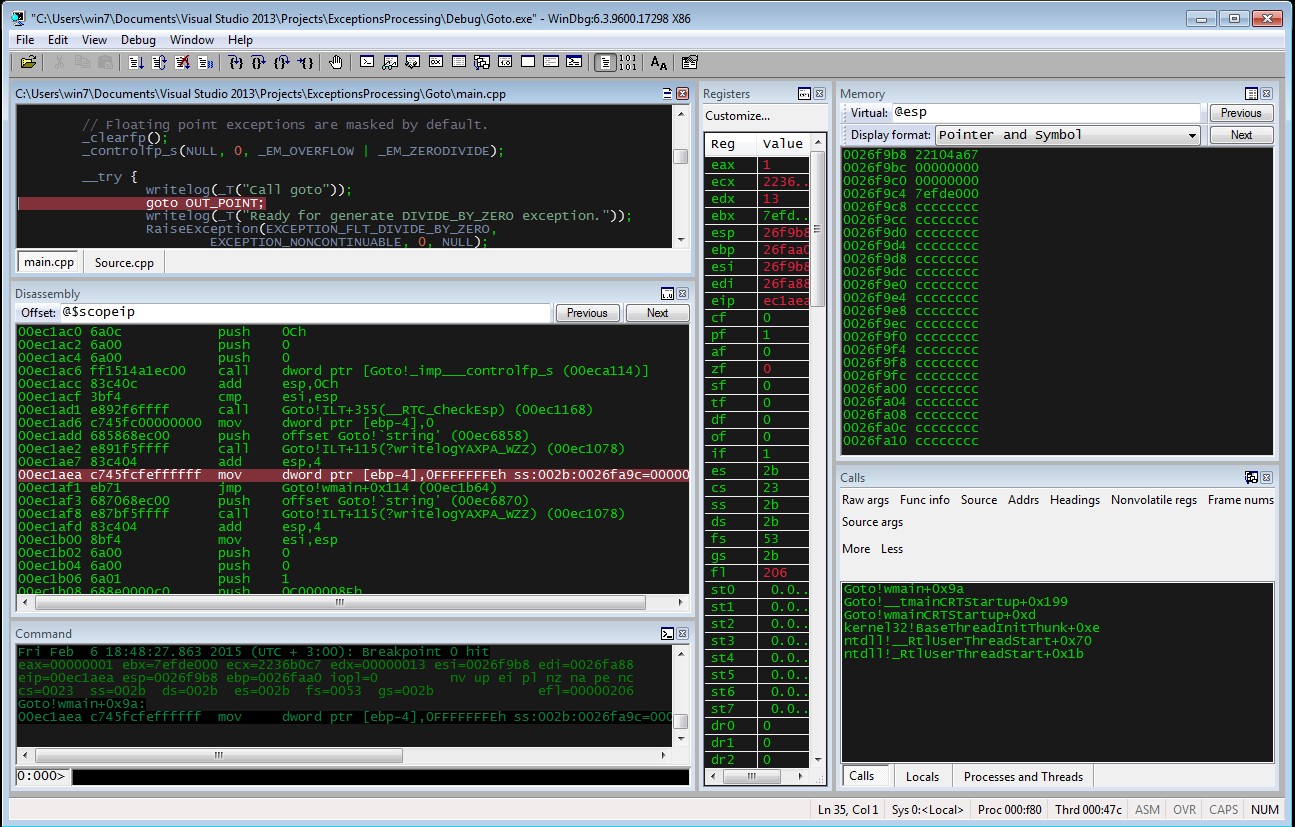
( LPCWSTR \*) pMessages , *// pointer to strings*

NULL )) { *// pointer to binary data buffer*

writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

}

}



## Рис. 17: Переход по goto

## На рисунке 17 видно, что как только достигнута строка с оператором goto, осуществляется безусловный переход к метке. Протокол работы программы (листинг 16) подтверждает, что до возбуждения исключения управление не дошло: после записи логе из 40-й строки идёт запись из 55-й, таким образом строки 44-46 пропущены.

## Листинг 16: Переход по оператору goto

1 [ 6 / 2 / 2015 18 : 52 : 53 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \ Exceptions Processing \ Debug \ Goto . exe is starting .

2 [ 6 / 2 / 2015 18 : 52 : 53 ] Call goto

3 [ 6 / 2 / 2015 18 : 52 : 53 ] A point outside the \_\_ try block .

4 [ 6 / 2 / 2015 18 : 52 : 53 ] Shutting down .

## Использование goto может привести к утечкам памяти в процессе раскрутки стека, в то же время он позволяет сделать переход сразу через несколько участков кода. Таким образом, сфера применения goto достаточно узкая, и требует достаточно чёткого понимания.

# Выход при помощи leave

## Листинг 17 похож на листинг 15, но за пределы охраняемого фрейма кода помогает выйти на этот раз leave. Оператор leave более эффективен, поскольку не вызывает разрушение стека.

## Листинг 17: Выход из блока охраняемого кода при помощи leave (src/ExceptionsProcessing/Leave/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

*/\**

*Task 8.*

*Get out of the \_\_ try block by using the leave;*

*\*/*

*//*

*//*

*//*

*IMPORTANT : Don ’ t forget to disable Enhanced Instructions !!!*

*Properties -> Configuration Properties -> C/ C++ -> Code Generation ->*

*Enable Enhanced Instruction Set*

*=*

*No*

*Enhanced*

*Instructions*

*(/ arch: IA32 )*

# include

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cstring >

< cfloat >

< excpt . h>

< windows . h>

< time . h>

# include

" messages . h"

*// log*

FILE \* logfile ; HANDLE eventlog ;

void void void void

initlog ( const \_ TCHAR \* prog );

closelog ();

writelog ( \_ TCHAR \* format , ...) ;

syslog ( WORD category , WORD identifier , LPWSTR message );

*// Defines the entry point for the console application .*

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// Init log*
3. initlog ( argv [0]) ;
4. eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 33

1. *// Floating point exceptions are masked by default .*
2. \_ clearfp ();
3. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 37

1. \_\_ try {
2. writelog (\_T(" Call

\_\_ leave "));

1. \_\_ leave ;
2. writelog (\_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
3. syslog ( ZERODIVIDE\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
4. \_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
5. Raise Exception ( EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ,
6. EXCEPTION\_ NONCONTINUABLE , 0 , NULL );
7. writelog (\_T(" DIVIDE\_ BY\_ ZERO exception is generated ."));

47 }

48 \_\_ except ( EXCEPTION\_ EXECUTE\_ HANDLER )

49 {

1. writelog (\_T(" Handler in action ."));
2. \_ tprintf (\_T(" Handler in action ."));
3. syslog ( ZERODIVIDE\_ CATEGORY , CAUGHT\_ EXCEPRION ,
4. \_T(" Handler in action ."));

54 }

1. writelog (\_T(" A point outside the

\_\_try block ."));

1. \_ tprintf (\_T(" A point outside the

\_\_try block ."));

1. syslog ( ZERODIVIDE\_ CATEGORY , CAUGHT\_ EXCEPRION ,
2. \_T(" A point outside the

### 59

\_\_try block ."));

1. closelog ();
2. Close Handle ( eventlog );
3. exit (0) ;

63 }

### 64

1. void initlog ( const \_ TCHAR \* prog ) {
2. \_ TCHAR logname [ 255 ];
3. wcscpy\_ s ( logname , prog );

### 68

1. *// replace extension*
2. \_ TCHAR \* extension ;
3. extension = wcsstr ( logname , \_T(". exe "));
4. wcsncpy\_ s ( extension , 5 , \_T(". log "), 4);

### 73

* 1. *// Try to open log file for append*
  2. if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {
  3. \_ wperror (\_T(" The following error occurred "));
  4. \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname );
  5. exit (1) ;

79 }

### 80

81 writelog (\_T("% s is starting ."), prog );

82 }

### 83

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

87 }

### 88

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;

### 92

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 95

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

### 100

1. *// Convert to normal representation .*
2. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
3. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
4. newtime . tm\_min , newtime . tm\_ sec );

### 105

1. *// Write date and time*
2. fwprintf ( logfile , \_T("% s"), buf );
3. *// Write all params*
4. va\_ start ( ap , format );
5. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);
6. fwprintf ( logfile , \_T("% s"), buf );
7. va\_ end ( ap);
8. *// New sting*
9. fwprintf ( logfile , \_T("\ n"));

### 115 }

### 116

1. void syslog ( WORD category , WORD identifier , LPWSTR message ) {
2. LPWSTR p Messages [1] = { message };

119

120

121

122

123

124

125

126

127

128

129

130

131

132

if (! Report Event (

eventlog ,

*// event*

EVENTLOG\_ INFORMATION\_ TYPE ,

category , identifier , NULL ,

1 ,

0 ,

*// event*

*log handle*

*// event type category*

*//*

*//*

*//*

*// event identifier*

*user security identifier number of substitution strings data size*

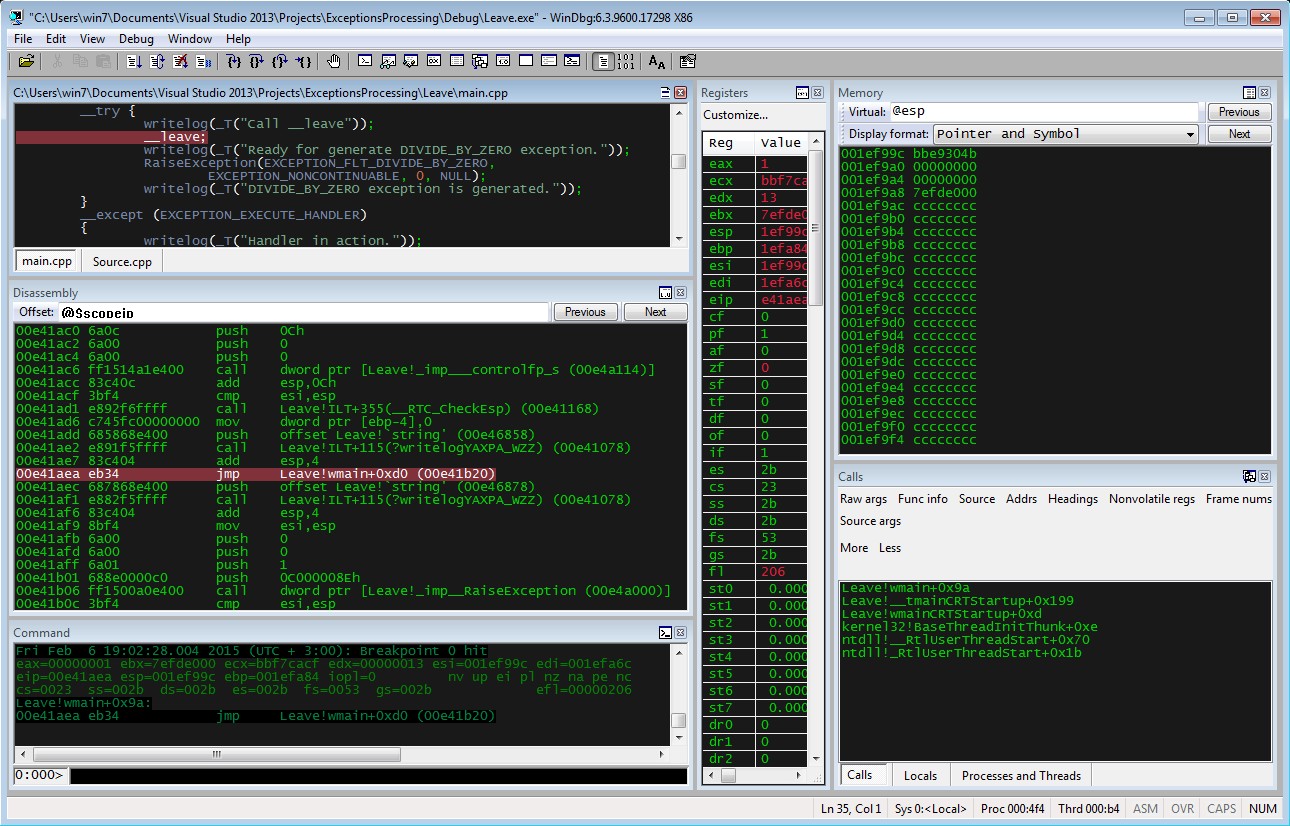
( LPCWSTR \*) pMessages , *// pointer to strings*

NULL )) { *// pointer to binary data buffer*

writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

}

}



## Рис. 18: Переход по leave Листинг 18: Переход по оператору leave

1 [ 6 / 2 / 2015 19 : 8 : 37 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \ Exceptions Processing \ Debug \ Leave . exe is starting .

2 [ 6 / 2 / 2015 19 : 8 : 37 ] Call \_\_ leave

3 [ 6 / 2 / 2015 19 : 8 : 37 ] A point outside the \_\_ try block .

4 [ 6 / 2 / 2015 19 : 8 : 38 ] Shutting down .

## Результат использования leave — переход в конец блока try (грубо говоря, это можно рассматривать это как goto переход на закрывающую фигурную скобку блока try и вход в блок finally естественным образом). По сути, результат прежний (если смотреть на листинг 18), но метод его достижения отличается – этот способ считается более правильным, т.к. не приводит к раскрутке стека.

## После перехода выполняется обработчик завершения. Хотя для получения того же резуль- тата можно использовать оператор goto, он (оператор goto) приводит к освобождению стека. Одним из применений этого оператора является трассировка программ.

# Преобразование SEH в C++ исключение

## Листинг 19 показывает встраивание SEH в механизм исключений C/С++. Для этого необходимо включить соответствующие опции в компиляторе (/EHa).

## Листинг 19: Трансформация исключений (src/ExceptionsProcessing/Translator/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

*/\**

*Task 9.*

*Convert structural exceptions to the C language exceptions , using the translator ;*

*\*/*

*//*

*//*

*//*

*IMPORTANT : Don ’ t forget to disable Enhanced Instructions !!!*

*Properties -> Configuration Properties -> C/ C++ -> Code Generation -> Enable Enhanced Instruction Set = No Enhanced Instructions (/ arch: IA32 )*

*//*

*//*

*//*

*IMPORTANT :*

*Properties Enable C++*

*Don ’ t forget to enable SEH !!!*

*-> Configuration Properties -> C/ C++ -> Code Generation -> Exceptions = Yes with SEH Exceptions (/ EHa)*

# include

# include

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cstring >

< cfloat >

< stdexcept >

< excpt . h>

< windows . h>

< time . h>

# include

" messages . h"

*// log*

FILE \* logfile ; HANDLE eventlog ;

1. void initlog ( const \_ TCHAR \* prog );
2. void closelog ();
3. void writelog ( \_ TCHAR \* format , ...) ;
4. void syslog ( WORD category , WORD identifier , LPWSTR message );

### 33

34 void translator ( unsigned int u, EXCEPTION\_ POINTERS \* pExp );

### 35

1. *// My exception*
2. class translator\_ exception {
3. public :
4. translator\_ exception ( const wchar\_ t \* str ) {
5. wcsncpy\_ s ( buf , sizeof ( buf ), str , sizeof ( buf ));

41 }

1. const wchar\_ t \* what () { return buf ; }
2. private :
3. wchar\_ t buf [ 255 ];

45 };

### 46

1. *// Defines the entry point for the console application .*
2. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
3. *// Init log*
4. initlog ( argv [0]) ;
5. eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 52

1. *// Floating point exceptions are masked by default .*
2. \_ clearfp ();
3. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 56

1. try {
2. writelog (\_T(" Ready for translator ativation ."));
3. \_ set\_ se\_ translator ( translator );
4. writelog (\_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
5. syslog ( ZERODIVIDE\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
6. \_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
7. Raise Exception ( EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ,
8. EXCEPTION\_ NONCONTINUABLE , 0 , NULL );
9. writelog (\_T(" DIVIDE\_ BY\_ ZERO exception is generated ."));

66 }

1. catch ( translator\_ exception & e) {
2. \_ tprintf (\_T(" CPP exception : % s"), e. what ());
3. writelog (\_T(" CPP exception : % s"), e. what ());
4. syslog ( OVERFLOW\_ CATEGORY , CAUGHT\_ EXCEPRION , \_T(" CPP exception "));

71 }

### 72

73 closelog ();

1. Close Handle ( eventlog );
2. exit (0) ;

76 }

### 77

1. void translator ( unsigned int u, EXCEPTION\_ POINTERS \* pExp ) {
2. writelog (\_T(" Translator in action ."));
3. if ( u == EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO )
4. throw translator\_ exception (\_T(" EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO "));

82 }

### 83

1. void initlog ( const \_ TCHAR \* prog ) {
2. \_ TCHAR logname [ 255 ];
3. wcscpy\_ s ( logname , prog );

### 87

1. *// replace extension*
2. \_ TCHAR \* extension ;
3. extension = wcsstr ( logname , \_T(". exe "));
4. wcsncpy\_ s ( extension , 5 , \_T(". log "), 4);

### 92

1. *// Try to open log file for append*
2. if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {
3. \_ wperror (\_T(" The following error occurred "));
4. \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname );
5. exit (1) ;

98 }

### 99

100 writelog (\_T("% s is starting ."), prog );

### 101 }

### 102

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

### 106 }

### 107

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;

### 111

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 114

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

|  |  |  |
| --- | --- | --- |
| 119 |  | |
| 120 |  | *// Convert to normal representation .* |
| 121 |  | swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday , |
| 122 |  | newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour , |
| 123  124 |  | newtime . tm\_min , newtime . tm\_ sec ); |
| 125 |  | *// Write date and time* |
| 126 |  | fwprintf ( logfile , \_T("% s"), buf ); |
| 127  128 |  | *// Write all params*  va\_ start ( ap , format ); |
| 129 |  | \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap); |
| 130 |  | fwprintf ( logfile , \_T("% s"), buf ); |
| 131 |  | va\_ end ( ap); |
| 132 |  | *// New sting* |
| 133 |  | fwprintf ( logfile , \_T("\ n")); |
| 134 | } |  |
| 135 |  |  |
| 136 | vo | id syslog ( WORD category , WORD identifier , LPWSTR message ) { |
| 137 |  | LPWSTR p Messages [1] = { message }; |
| 138 |  |  |
| 139 |  | if (! Report Event ( |
| 140 |  | eventlog , *// event log handle* |

## На рисунке 19 видна передача управления от генерации исключения в ядре к созданию поль- зовательского исключения. Листинг 20 показывает, какие участи кода были задействованы и в каком порядке.

141

142

143

144

145

146

147

148

149

150

151

EVENTLOG\_ INFORMATION\_ TYPE , *// event type*

category ,

identifier , NULL ,

1 ,

0 ,

*//*

*//*

*//*

*// event category*

*// event identifier user security identifier*

*number of substitution strings*

*data size*

( LPCWSTR \*) pMessages ,

*// pointer to strings*

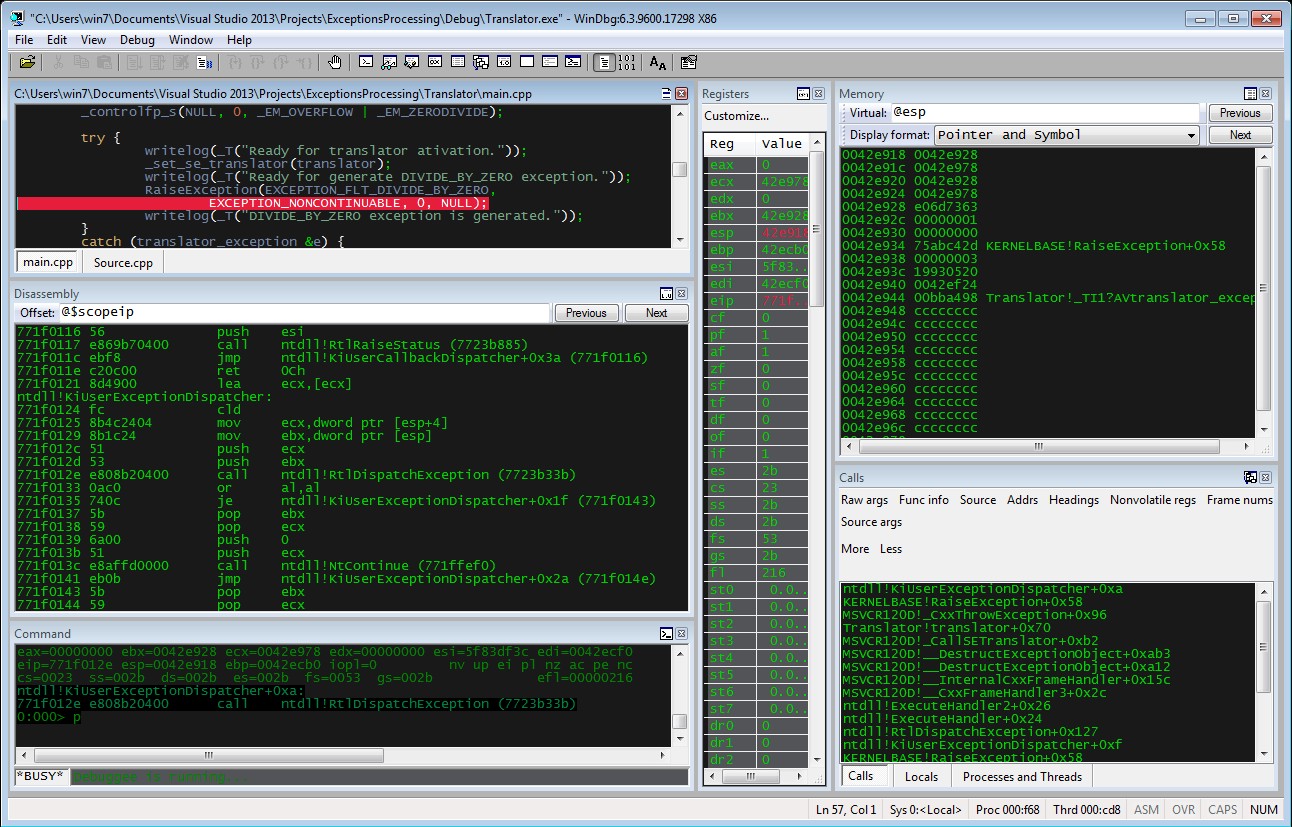
NULL )) {

*// pointer to binary data buffer*

writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

}

}



## Рис. 19: Передача управления коду создания пользовательского исключения Листинг 20: Результат работы Translator.exe

1. [ 6 / 2 / 2015 19 : 33 : 15 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \ Exceptions Processing \ Debug \ Translator . exe is starting .
2. [ 6 / 2 / 2015 19 : 33 : 15 ] Ready for translator ativation .
3. [ 6 / 2 / 2015 19 : 33 : 15 ] Ready for generate DIVIDE\_ BY\_ ZERO exception .

4 [ 6 / 2 / 2015 19 : 33 : 15 ] Translator in action .

5 [ 6 / 2 / 2015 19 : 33 : 15 ] CPP exception : EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO

6 [ 6 / 2 / 2015 19 : 33 : 15 ] Shutting down .

## Если проследить за передачей управления по стеку вызовов, то сразу после возбуждения исключения в 63-й строке, управление передаётся транслятору, где генерируется привычное С++-исключения (в данном случае используется собственный класс исключения, опре- делённый в 37-й строке), и только после этого в блок catch, где происходит обработка исключения.

## Этот механизм способен обеспечить взаимодействие SEH с другими языками и системами.

# Финальный обработчик finally

## В листинге 21 исключение как таковое отсутствует, но есть охраняемый блок кода, и блок

## finally, управление в который будет передано в любой ситуации.

## Листинг 21: Исполнение кода в блоке finally (src/ExceptionsProcessing/Finally/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

*/\**

*Task 10.*

*Use the final handler finally;*

*\*/*

*//*

*//*

*//*

*IMPORTANT : Don ’ t forget to disable Enhanced Instructions !!!*

*Properties -> Configuration Properties -> C/ C++ -> Code Generation ->*

*Enable Enhanced Instruction Set = No Enhanced Instructions*

*(/ arch: IA32 )*

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cfloat >

< excpt . h>

< time . h>

< windows . h>

# include

" messages . h"

*// log*

FILE \* logfile ; HANDLE eventlog ;

void void void void

initlog ( const \_ TCHAR \* prog );

closelog ();

writelog ( \_ TCHAR \* format , ...) ;

syslog ( WORD category , WORD identifier , LPWSTR message );

*// Defines the entry point for the console application .*

int \_ tmain ( int argc , \_ TCHAR \* argv []) {

*// Init log*

initlog ( argv [0]) ;

31 eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 32

1. *// Floating point exceptions are masked by default .*
2. \_ clearfp ();
3. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 36

1. \_\_ try {
2. *// No exception*
3. writelog (\_T(" Try block ."));

40 }

41 \_\_ finally

42 {

1. writelog (\_T(" Thre is no exception , but the handler is called ."));
2. \_ tprintf (\_T(" Thre is no exception , but the handler is called ."));
3. syslog ( OVERFLOW\_ CATEGORY , CAUGHT\_ EXCEPRION ,
4. \_T(" Thre is no exception , but the handler is called ."));

47 }

### 48

1. closelog ();
2. Close Handle ( eventlog );
3. exit (0) ;

52 }

### 53

1. void initlog ( const \_ TCHAR \* prog ) {
2. \_ TCHAR logname [ 255 ];
3. wcscpy\_ s ( logname , prog );

### 57

1. *// replace extension*
2. \_ TCHAR \* extension ;
3. extension = wcsstr ( logname , \_T(". exe "));
4. wcsncpy\_ s ( extension , 5 , \_T(". log "), 4);

### 62

1. *// Try to open log file for append*
2. if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {
3. \_ wperror (\_T(" The following error occurred "));
4. \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname );
5. exit (1) ;

68 }

### 69

70 writelog (\_T("% s is starting ."), prog );

71 }

### 72

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

76 }

### 77

* 1. void writelog ( \_ TCHAR \* format , ...) {
  2. \_ TCHAR buf [ 255 ];
  3. va\_ list ap;

### 81

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 84

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

### 89

1. *// Convert to normal representation .*
2. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
3. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
4. newtime . tm\_min , newtime . tm\_ sec );

### 94

1. *// Write date and time*
2. fwprintf ( logfile , \_T("% s"), buf );
3. *// Write all params*
4. va\_ start ( ap , format );
5. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);
6. fwprintf ( logfile , \_T("% s"), buf );
7. va\_ end ( ap);
8. *// New sting*
9. fwprintf ( logfile , \_T("\ n"));

### 104 }

### 105

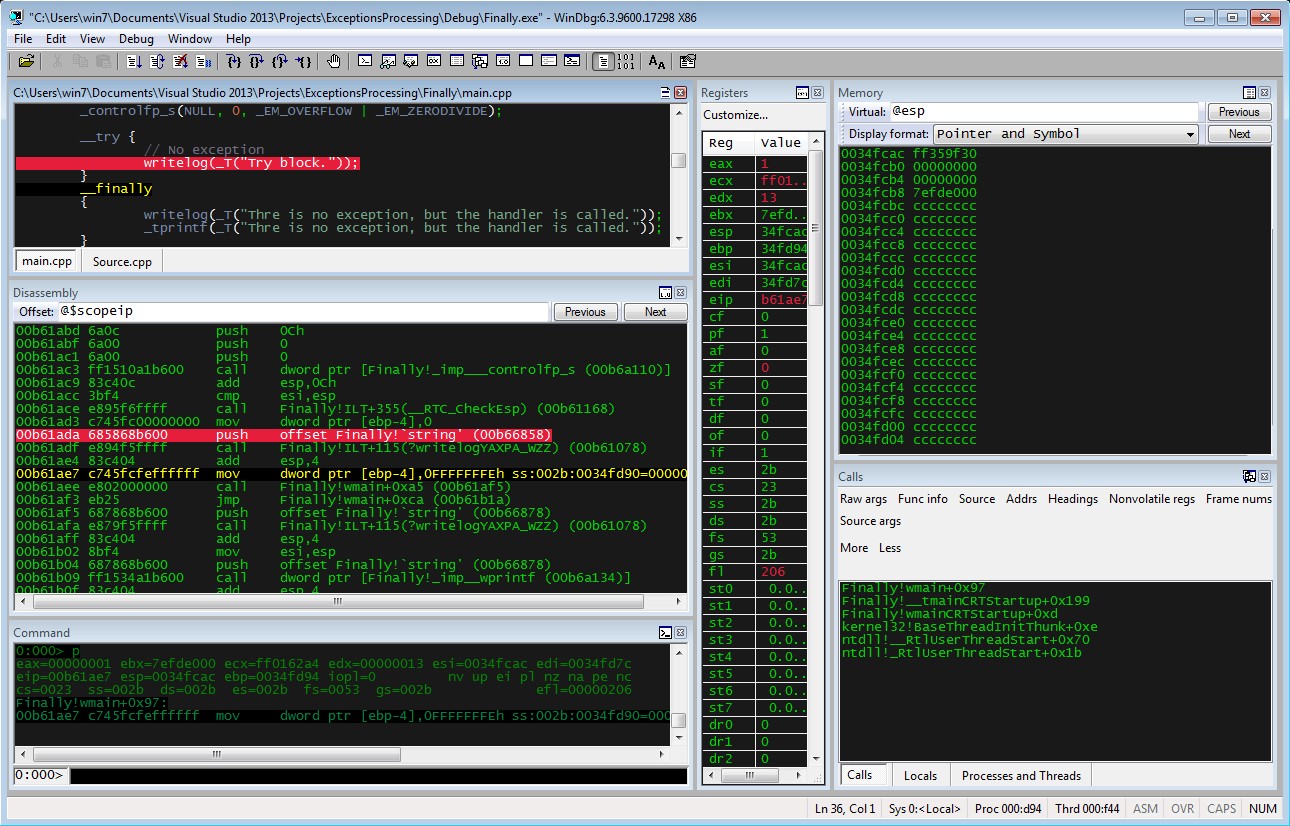
1. void syslog ( WORD category , WORD identifier , LPWSTR message ) {
2. LPWSTR p Messages [1] = { message };

### 108

1. if (! Report Event (
2. eventlog , *// event log handle*
3. EVENTLOG\_ INFORMATION\_ TYPE , *// event type*
4. category , *// event category*
5. identifier , *// event identifier*
6. NULL , *// user security identifier*
7. 1 , *// number of substitution strings*
8. 0 , *// data size*
9. ( LPCWSTR \*) pMessages , *// pointer to strings*
10. NULL )) { *// pointer to binary data buffer*
11. writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

### 120 }

121 }



## Рис. 20: Переход в блок finally Листинг 22: Результат работы Finally.exe

1 [ 6 / 2 / 2015 19 : 42 : 27 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \ Exceptions Processing \ Debug \ Finally . exe is starting .

2 [ 6 / 2 / 2015 19 : 42 : 27 ] Try block .

3 [ 6 / 2 / 2015 19 : 42 : 27 ] Thre is no exception , but the handler is called .

4 [ 6 / 2 / 2015 19 : 42 : 27 ] Shutting down .

## Вместо передачи управления обратно в программу, управление передаётся в блок finally (рисунок 20). Более того, управление туда будет передано даже если блок защищаемого кода будет пуст. Листинг 22 показывает порядок исполнения кода.

## Похожие механизмы есть в других распространённых языках программирования, они позволяют обеспечить строгие гарантии исключений, и не допустить нахождение объекта в не консистентном состоянии.

# Использование функции AbnormalTermination

## В листинге 23 сравниваются два механизма из блока try. Благодаря тому, что управление будет передано блоку finally в любом случае, оказывается удобно в этом блоке проверять корректность выхода из блока try (при помощи функции AbnormalTermination), и, в случае необходимости, корректно освобождать захваченные ресурсы.

## Листинг 23: Проверка корректности выхода из блока try (src/ExceptionsProcessing/AbnormalTermination/main.cpp)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

*/\**

*Task 11.*

*Check the correctness of the exit from the \_\_try block using the Abnormal Termination function in the final handler finally.*

*\*/*

*//*

*//*

*//*

*IMPORTANT : Don ’ t forget to disable Enhanced Instructions !!!*

*Properties -> Configuration Properties -> C/ C++ -> Code Generation -> Enable Enhanced Instruction Set = No Enhanced Instructions (/ arch: IA32 )*

# include

# include

# include

# include

# include

# include

# include

< stdio . h>

< tchar . h>

< cstring >

< cfloat >

< excpt . h>

< windows . h>

< time . h>

# include

" messages . h"

*// log*

FILE \* logfile ; HANDLE eventlog ;

void initlog ( const \_ TCHAR \* prog );

1. void closelog ();
2. void writelog ( \_ TCHAR \* format , ...) ;
3. void syslog ( WORD category , WORD identifier , LPWSTR message );

### 28

1. *// Defines the entry point for the console application .*
2. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
3. *// Init log*
4. initlog ( argv [0]) ;
5. eventlog = Register Event Source ( NULL , L" My Event Provider ");

### 34

1. *// Floating point exceptions are masked by default .*
2. \_ clearfp ();
3. \_ controlfp\_ s ( NULL , 0 , \_ EM\_ OVERFLOW | \_ EM\_ ZERODIVIDE );

### 38

1. \_\_ try {
2. writelog (\_T(" Call goto "));
3. goto OUT\_ POINT ;
4. writelog (\_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
5. syslog ( ZERODIVIDE\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
6. \_T(" Ready for generate DIVIDE\_ BY\_ ZERO exception ."));
7. Raise Exception ( EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ,
8. EXCEPTION\_ NONCONTINUABLE , 0 , NULL );
9. writelog (\_T(" DIVIDE\_ BY\_ ZERO exception is generated ."));

48 }

49 \_\_ finally

50 {

1. if ( Abnormal Termination ()) {
2. writelog (\_T("% s"), \_T(" Abnormal termination in goto case ."));
3. \_ tprintf (\_T("% s"), \_T(" Abnormal termination in goto case .\ n"));
4. syslog ( ZERODIVIDE\_ CATEGORY , CAUGHT\_ EXCEPRION ,
5. \_T(" Abnormal termination in goto case ."));

56 }

1. else {
2. writelog (\_T("% s"), \_T(" Normal termination in goto case ."));
3. \_ tprintf (\_T("% s"), \_T(" Normal termination in goto case .\ n"));
4. syslog ( ZERODIVIDE\_ CATEGORY , CAUGHT\_ EXCEPRION ,
5. \_T(" Normal termination in goto case ."));

62 }

63 }

1. OUT\_ POINT :
2. writelog (\_T(" A point outside the first

### 66

\_\_ try block ."));

1. \_\_ try {
2. writelog (\_T(" Call

\_\_ leave "));

1. \_\_ leave ;
2. writelog (\_T(" Ready for generate EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO exception ."

));

1. syslog ( OVERFLOW\_ CATEGORY , READY\_ FOR\_ EXCEPTION ,
2. \_T(" Ready for generate FLT\_ OVERFLOW exception ."));
3. Raise Exception ( EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO ,
4. EXCEPTION\_ NONCONTINUABLE , 0 , NULL );
5. writelog (\_T(" EXCEPTION\_ FLT\_ DIVIDE\_ BY\_ ZERO exception is generated ."));

76 }

77 \_\_ finally

78 {

1. if ( Abnormal Termination ()) {
2. writelog (\_T("% s"),

\_T(" Abnormal termination in

\_\_ leave case ."));

1. \_ tprintf (\_T("% s"),

\_T(" Abnormal termination in

\_\_ leave case .\ n"));

1. syslog ( OVERFLOW\_ CATEGORY , CAUGHT\_ EXCEPRION ,

### 83

84 }

\_T(" Abnormal termination in

\_\_ leave case ."));

1. else {
2. writelog (\_T("% s"),

\_T(" Normal termination in

\_\_ leave case ."));

1. \_ tprintf (\_T("% s"),

\_T(" Normal termination in

\_\_ leave case .\ n"));

1. syslog ( OVERFLOW\_ CATEGORY , CAUGHT\_ EXCEPRION ,

### 89

90 }

91 }

\_T(" Normal termination in

\_\_ leave case ."));

92 writelog (\_T(" A point outside the second

### 93

1. closelog ();
2. Close Handle ( eventlog );
3. exit (0) ;

97 }

### 98

99 void initlog ( const \_ TCHAR \* prog ) {

1. \_ TCHAR logname [ 255 ];
2. wcscpy\_ s ( logname , prog );

### 102

1. *// replace extension*
2. \_ TCHAR \* extension ;
3. extension = wcsstr ( logname , \_T(". exe "));
4. wcsncpy\_ s ( extension , 5 , \_T(". log "), 4);

### 107

1. *// Try to open log file for append*

\_\_ try block ."));

1. if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {
2. \_ wperror (\_T(" The following error occurred "));
3. \_ tprintf (\_T(" Can ’t open log file % s\ n"), logname );
4. exit (1) ;

### 113 }

### 114

115 writelog (\_T("% s is starting ."), prog );

### 116 }

### 117

1. void closelog () {
2. writelog (\_T(" Shutting down .\ n"));
3. fclose ( logfile );

### 121 }

### 122

1. void writelog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;

### 126

1. struct tm newtime ;
2. \_\_ time 64 \_ t long\_ time ;

### 129

1. *// Get time as 64 - bit integer .*
2. \_ time 64 (& long\_ time );
3. *// Convert to local time.*
4. \_ localtime 64 \_ s (& newtime , & long\_ time );

### 134

1. *// Convert to normal representation .*
2. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
3. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
4. newtime . tm\_min , newtime . tm\_ sec );

### 139

1. *// Write date and time*
2. fwprintf ( logfile , \_T("% s"), buf );
3. *// Write all params*
4. va\_ start ( ap , format );
5. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);
6. fwprintf ( logfile , \_T("% s"), buf );
7. va\_ end ( ap);
8. *// New sting*
9. fwprintf ( logfile , \_T("\ n"));

### 149 }

### 150

1. void syslog ( WORD category , WORD identifier , LPWSTR message ) {
2. LPWSTR p Messages [1] = { message };

### 153

1. if (! Report Event (
2. eventlog , *// event log handle*
3. EVENTLOG\_ INFORMATION\_ TYPE , *// event type*
4. category , *// event category*
5. identifier , *// event identifier*

162

163

164

165

166

( LPCWSTR \*) pMessages ,

*// pointer to strings*

NULL )) {

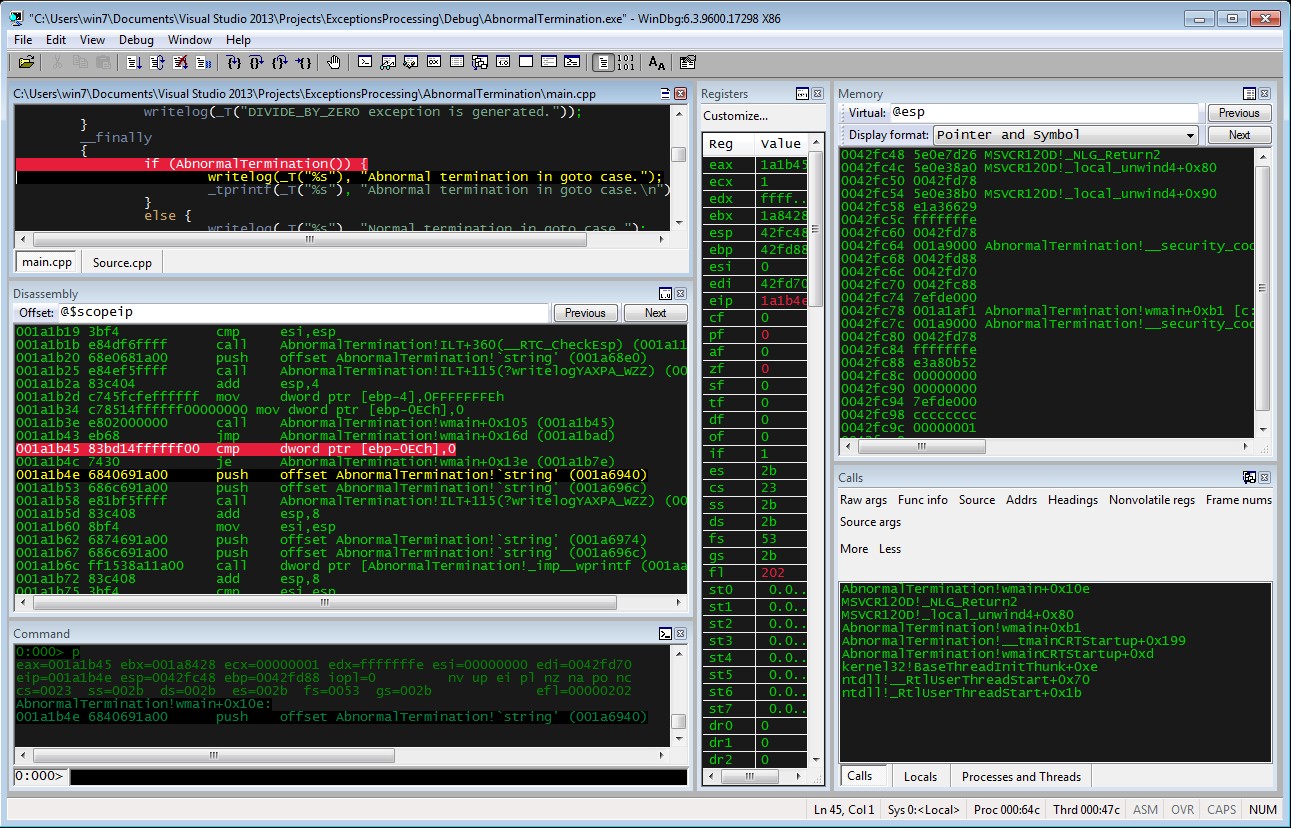
*// pointer to binary data buffer*

writelog (\_T(" Report Event failed with 0 x% x"), Get Last Error ());

}

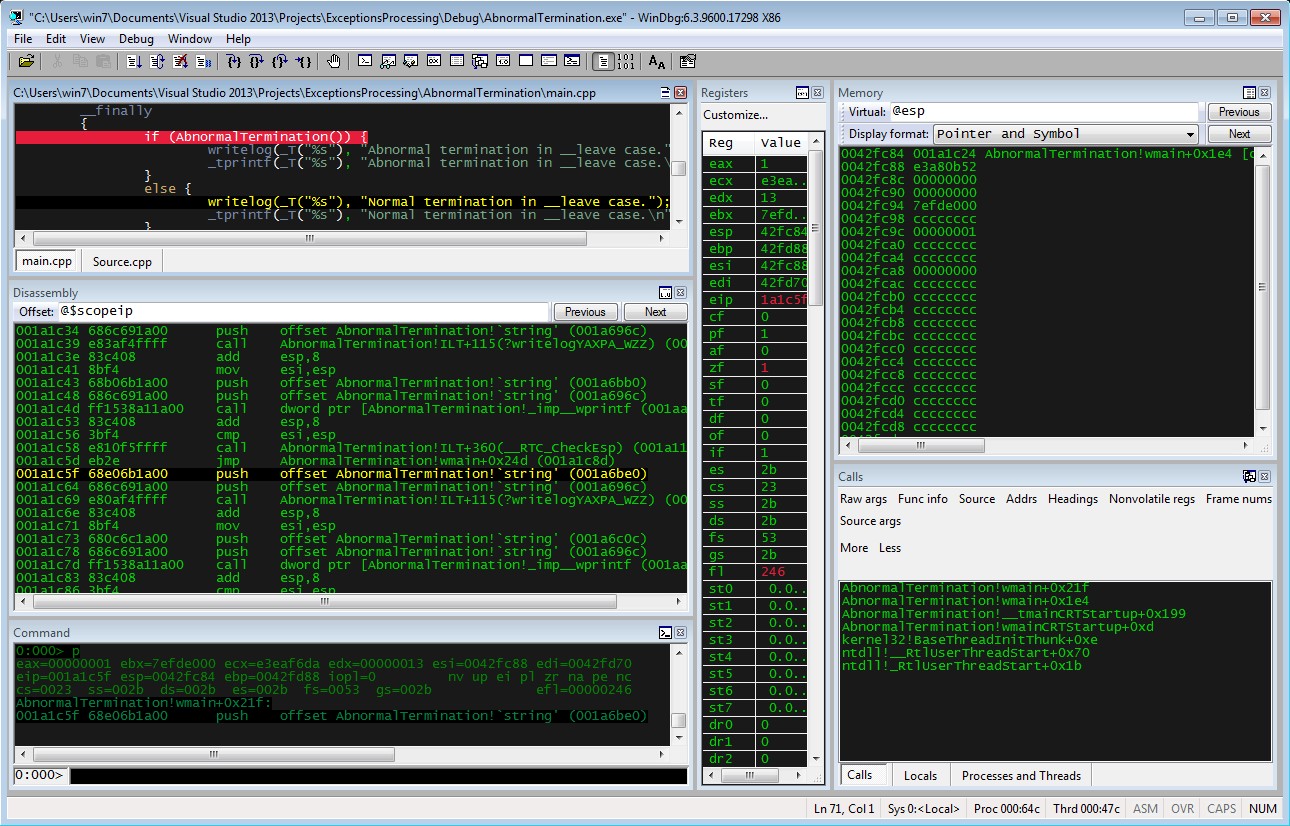
}

|  |  |  |  |
| --- | --- | --- | --- |
| 159 | NULL , | *//* | *user security identifier* |
| 160 | 1 , | *//* | *number of substitution strings* |
| 161 | 0 , | *//* | *data size* |



## Рис. 21: Выход из защищаемого блока по goto

## Функция AbnormalTermination() позволяет определить на сколько правильным был выход из защищаемого кода в случае с goto (рисунок 21) и leave (рисунок 22). Протокол работы программы представлен в листинге 22.



## Рис. 22: Выход из защищаемого блока по leave

## В зависимости от этого принимается решение об освобождении захваченных ресурсов, но если по какой-то причине нужно выйти из защищаемого блока (хотя причина такой необходимости не очевидна) лучше использовать leave, т.к. с goto больше шансов на утечку ресурсов, захваченных (и не освобождённых) в блоке \_\_try.

## Листинг 24: Результат работы Finally.exe

1 [ 6 / 2 / 2015 19 : 58 : 19 ] C:\ Users \ win7 \ Documents \ Visual Studio 2013 \ Projects \ Exceptions Processing \ Debug \ Abnormal Termination . exe is starting .

2 [ 6 / 2 / 2015 19 : 58 : 19 ] Call goto

1. [ 6 / 2 / 2015 19 : 58 : 19 ] Abnormal termination in goto case .
2. [ 6 / 2 / 2015 19 : 58 : 19 ] A point outside the first \_\_try block .

5 [ 6 / 2 / 2015 19 : 58 : 19 ] Call \_\_ leave

1. [ 6 / 2 / 2015 19 : 58 : 19 ] Normal termination in \_\_ leave case .
2. [ 6 / 2 / 2015 19 : 58 : 19 ] A point outside the second \_\_try block .

8 [ 6 / 2 / 2015 19 : 58 : 19 ] Shutting down .

# Заключение

## При обработке исключений в С++ используются ключевые слова catch и throw, а сам меха- низм исключений реализован с использованием SEH. Тем не менее, обработка исключений в С++ и SEH — это разные вещи. Их совместное применение требует внимательного обра- щения, поскольку обработчики исключений, написанные пользователем и сгенерированные C++, могут взаимодействовать между собой и приводить к нежелательным последствиям. Документация Microsoft рекомендует полностью отказаться от использования обработчиков Windows в прикладных программах на С++ и ограничиться применением в них только обработчиков исключений С++.

## Кроме того, обработчики исключений или завершения Windows не осуществляют вызов деструкторов, что в ряде случаев необходимо для уничтожения экземпляров объектов С++.

## В то же время, наличие таких мощных инструментов как блок finally, гибкая система фильтрации и извлечение контекста исключения делает их незаменимыми при разработке системного ПО.

## Таким образом, нужно чётко понимать, что механизм SEH и исключения, реализованные на уровня языка C++ это разные инструменты, требующие разного подхода.