

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
Введение	4
Исключения с помощью WinAPI	14
Использование GetExceptionCode	24
Пользовательская функция-фильтр	30
Использование RaiseException	37
Необрабатываемые исключения	44
Вложенные исключения	52
Выход при помощи goto	57
Выход при помощи __leave	62
Преобразование SEH в C++ исключение	67
Финальный обработчик finally	72
Использование функции AbnormalTermination	76
Заключение	82
Список литературы	83

Постановка задачи

1. Сгенерировать и обработать исключения с помощью функций WinAPI;
2. Получить код исключения с помощью функции `GetExceptionCode`.
 - Использовать эту функции в выражении фильтре;
 - Использовать эту функцию в обработчике.
3. Создать собственную функцию-фильтр;
4. Получить информацию об исключении с помощью функции `GetExceptionInformation`; сгенерировать исключение с помощью функции `RaiseException`;
5. Использовать функции `UnhandledExceptionFilter` и `SetUnhandledExceptionFilter` для необработанных исключений;
6. Обработать вложенные исключения;
7. Выйти из блока `__try` с помощью оператора `goto`;
8. Выйти из блока `__try` с помощью оператора `__leave`;
9. Преобразовать структурное исключение в исключение языка C, используя функцию `translator`;
10. Использовать финальный обработчик `finally`;
11. 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 аппаратных исключений является SEH (Structured Exception Handling), позволяющий программистам контролировать обработку исключений, а также являющийся отладочным средством[1].

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 стандартным 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[2]. 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" Панель управления "Система и безопасность" Администрирование затем дважды щелкнуть "Просмотр событий" (окно просмотра событий показано на рисунке 1). Выбрать интересующий 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.

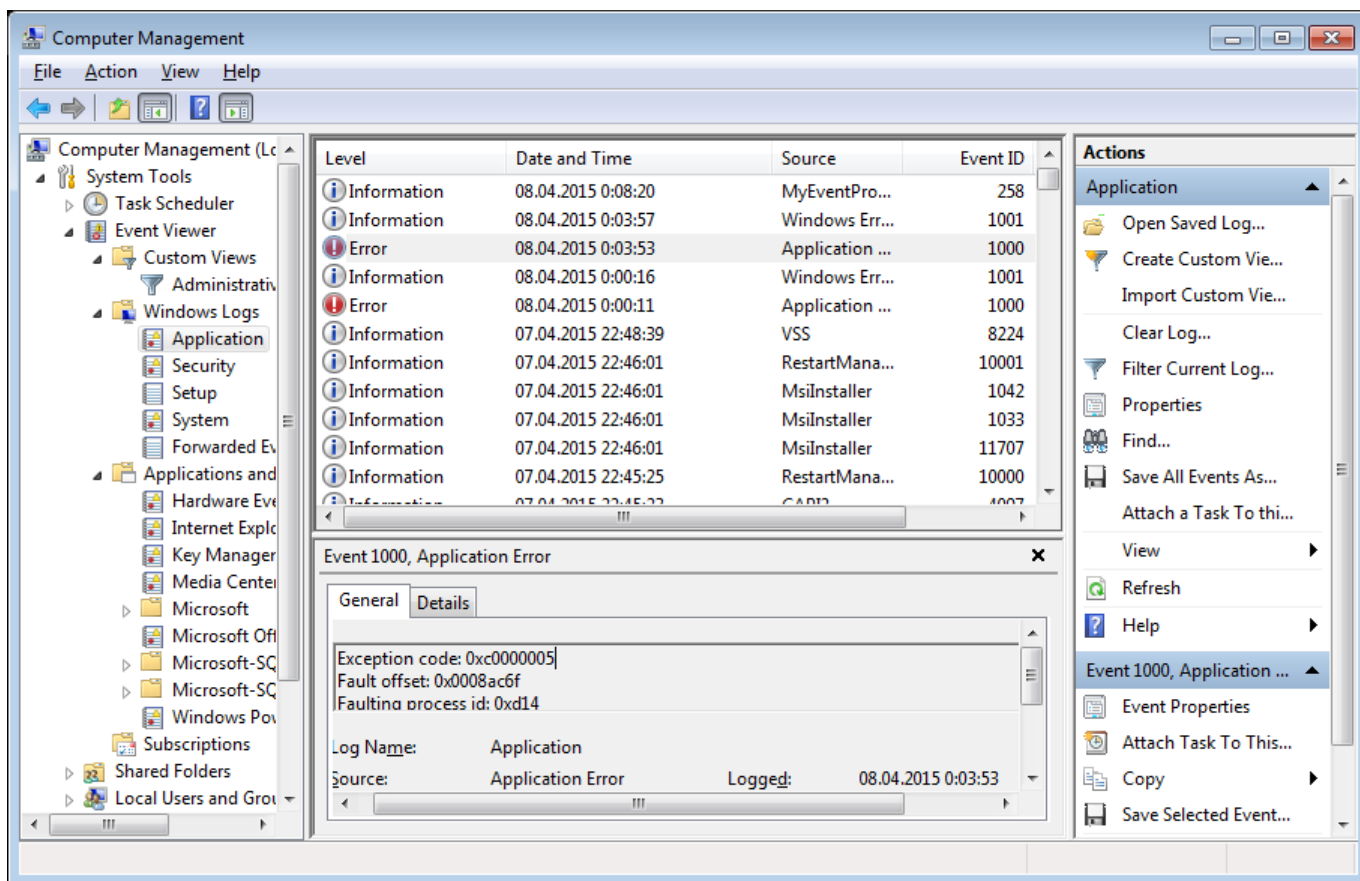


Рис. 1: Просмотру событий системного журнала Windows.

- 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 переполнения 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 LanguageNames=(Russian=0x419:MSG00419)
3 LanguageNames=(English=0x409:MSG00409)
4
5 ;// Categories
6
7 MessageIdTypedef=WORD
8
9 MessageId=0x1
10 SymbolicName=OVERFLOW_CATEGORY
11 Language=English
12 An overflow exception category.
13 .
14
15 Language=Russian
16 События переполнения
17 .
18
19 MessageId=0x2
20 SymbolicName=ZERODIVIDE_CATEGORY
21 Language=English
22 A division by zero exception category.
23 .
24
25 Language=Russian
26 События деления на 0
27 .
28
29 ;// Determiners
30
31 MessageIdTypedef=DWORD
32
33 MessageId=0x100
34 SymbolicName=READY_FOR_EXCEPTION
35 Language=English
36 Ready for generate exception.
37 .
38
39 Language=Russian
40 Готовность приложения сгенерировать исключительное событие.
41 .
42
43 MessageId=0x101
44 SymbolicName=CAUGHT_EXCEPRION
```

```
45 Language=English
46 Exclusive event happened.
47 .
48
49 Language=Russian
50 Произошло (и поймано) исключительное событие.
51 .
```

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

```
mc -U messages.mc
```

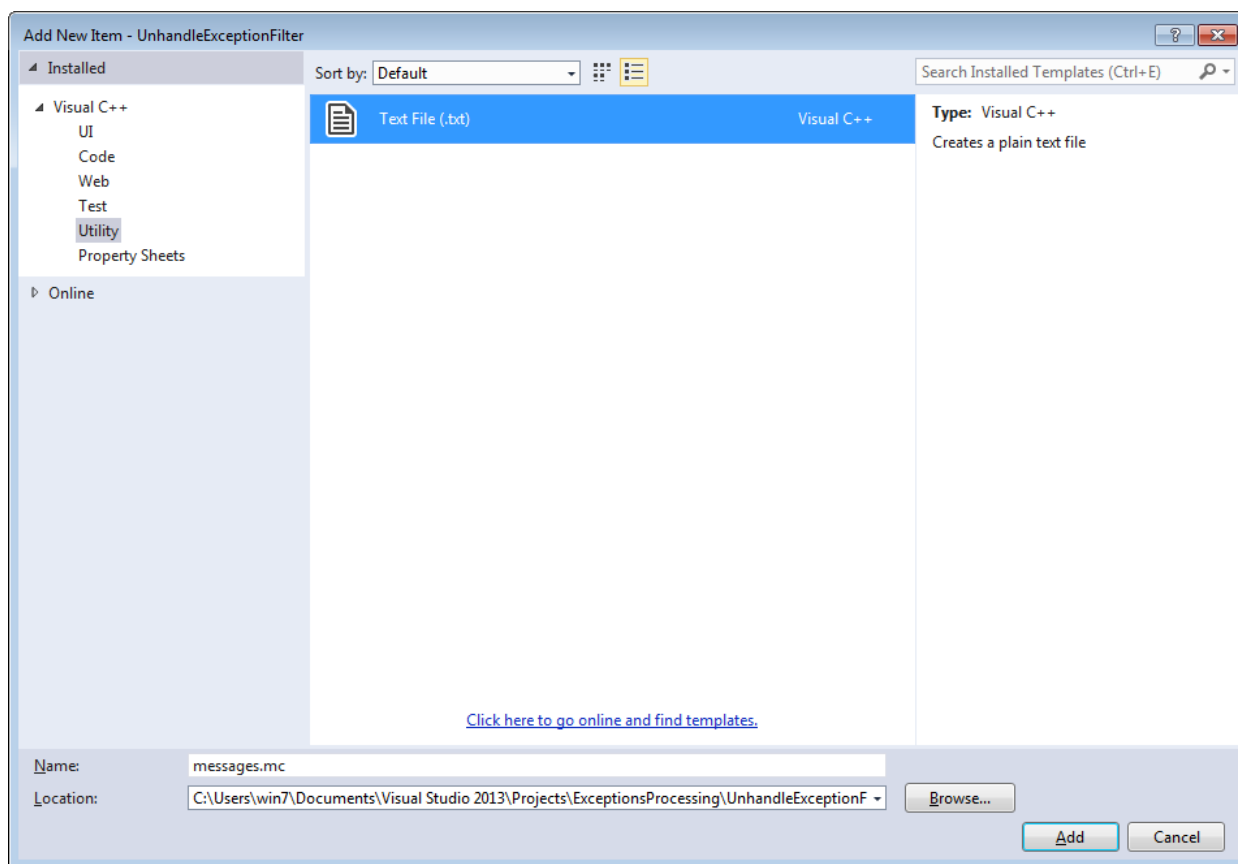


Рис. 2: Создание скрипта для генерации ресурсов в проекте.

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 "Настраиваемый инструмент построения" где нужно выбрать следующие параметры (смотри рисунок 4):

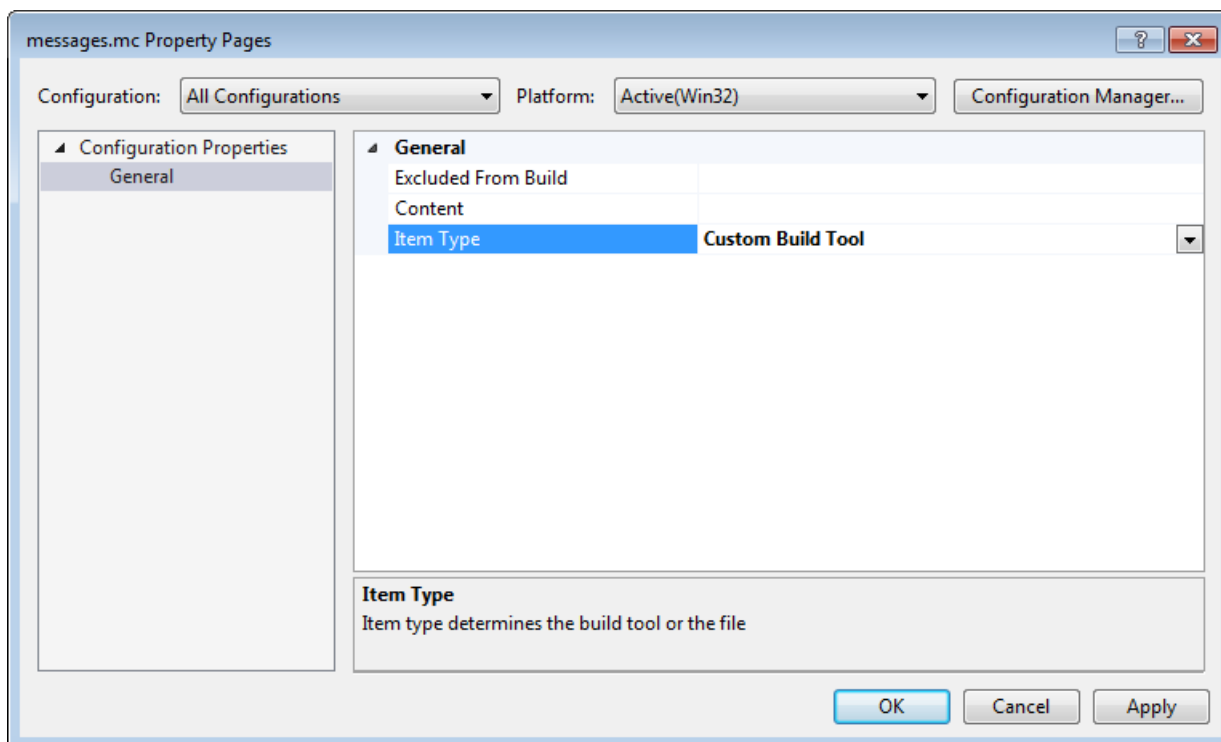


Рис. 3: Выбор типа элемента для файла messages.mc

Командная строка: `mc "%(FullPath)"`

Описание: `Compiling Messages...`

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

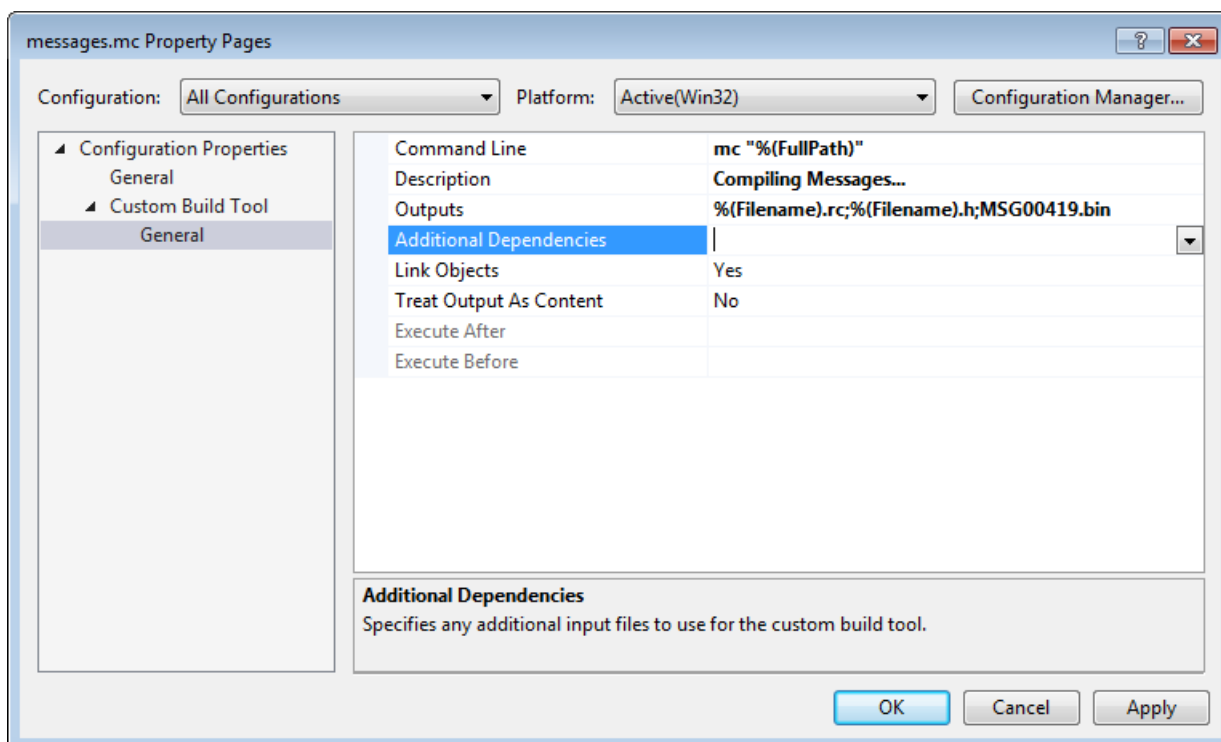


Рис. 4: Настройки исполнения скрипта генерации ресурсов

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 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 // Language
2 // Tategories
3 //
4 // Values are 32 bit values laid out as follows:
5 //
6 //   3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
7 //   1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
8 //   +---+---+---+---+---+---+---+---+---+---+---+---+
9 //   |Sev|C|R|      Facility      |      Code      |
10 //   +---+---+---+---+---+---+---+---+---+---+---+---+
11 //
12 // where
13 //
14 //     Sev - is the severity code
15 //
16 //         00 - Success
17 //         01 - Informational
18 //         10 - Warning
19 //         11 - Error
20 //
21 //     C - is the Customer code flag
22 //
23 //     R - is a reserved bit
24 //
25 //     Facility - is the facility code
26 //
27 //     Code - is the facility's status code
28 //
29 //
```

```

30 // Define the facility codes
31 //
32
33
34 //
35 // Define the severity codes
36 //
37
38
39 //
40 // MessageId: OVERFLOW_CATEGORY
41 //
42 // MessageText:
43 //
44 // An overflow exception category.
45 //
46 #define OVERFLOW_CATEGORY ((WORD)0x00000001L)
47
48 //
49 // MessageId: ZERODIVIDE_CATEGORY
50 //
51 // MessageText:
52 //
53 // A division by zero exception category.
54 //
55 #define ZERODIVIDE_CATEGORY ((WORD)0x00000002L)
56
57 // Determiners
58 //
59 // MessageId: READY_FOR_EXCEPTION
60 //
61 // MessageText:
62 //
63 // Ready for generate exception.
64 //
65 #define READY_FOR_EXCEPTION ((DWORD)0x00000100L)
66
67 //
68 // MessageId: CAUGHT_EXCEPRION
69 //
70 // MessageText:
71 //
72 // Exclusive event happened.
73 //
74 #define CAUGHT_EXCEPRION ((DWORD)0x00000101L)

```

После этого с системным журналом уже можно работать, но каждое событие будет начинаться 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 удастся 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 зарегистрировать 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 команда

```
link /DLL /NOENTRY messages.res
```

После выполнения этой команды будет создан файл messages.dll.

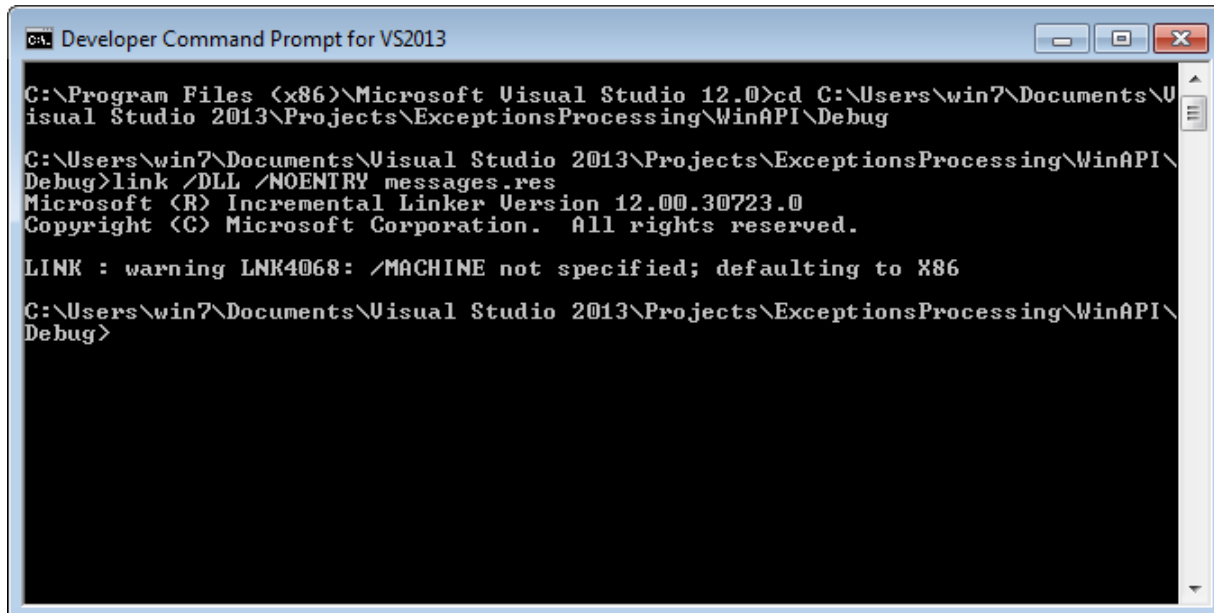


Рис. 5: Генерация библиотеки ресурсов

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, перечисленные 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.

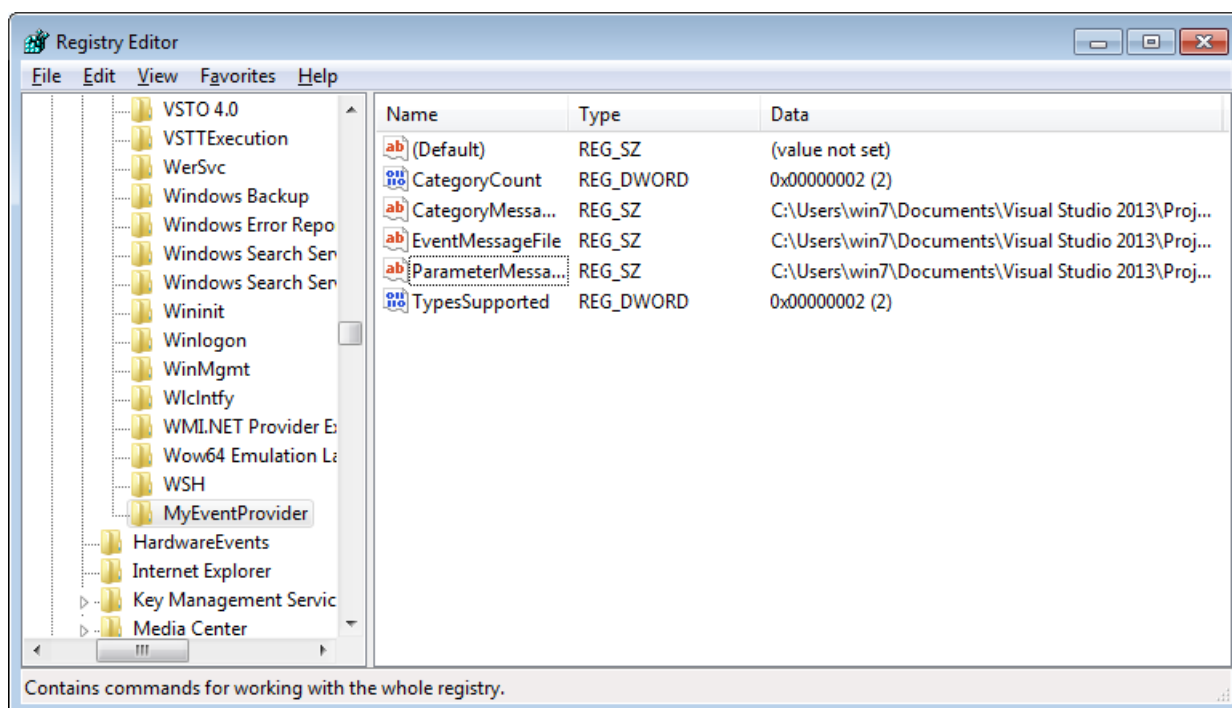


Рис. 6: Редактирование системного реестра

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 функций WinAPI.

В листинге 3 показана работа с исключениями[1]. 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  /* Task 1.
2  Generate and handle exceptions using the WinAPI functions;
3  */
4
5  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!
6  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
7  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)
8
9  #include <stdio.h>
10 #include <tchar.h>
11 #include <cstring>
12 #include <cfloat>
13 #include <cmath>
14 #include <excpt.h>
15 #include <windows.h>
16 #include <time.h>
```

```

17
18 #include "messages.h"
19
20 // log
21 FILE* logfile;
22 HANDLE eventlog;
23
24 void usage(const _TCHAR* prog);
25 void initlog(const _TCHAR* prog);
26 void closelog();
27 void writelog(_TCHAR* format, ...);
28 void syslog(WORD category, WORD identifier, LPWSTR message);
29
30 // Task switcher
31 enum {
32     DIVIDE_BY_ZERO,
33     FLT_OVERFLOW
34 } task;
35
36 // Defines the entry point for the console application.
37 int _tmain(int argc, _TCHAR* argv[]) {
38     //Init log
39     initlog(argv[0]);
40     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
41
42     // Check parameters number
43     if (argc != 2) {
44         _tprintf(_T("Too few parameters.\n\n"));
45         writelog(_T("Too few parameters."));
46         usage(argv[0]);
47         closelog();
48         exit(1);
49     }
50
51     // Set task
52     if (!_tcscmp(_T("-d"), argv[1])) {
53         task = DIVIDE_BY_ZERO;
54         writelog(_T("Task: DIVIDE_BY_ZERO exception."));
55     }
56     else if (!_tcscmp(_T("-o"), argv[1])) {
57         task = FLT_OVERFLOW;
58         writelog(_T("Task: FLT_OVERFLOW exception."));
59     }
60     else {
61         _tprintf(_T("Can't parse parameters.\n\n"));

```

```

62     writelog(_T("Can't parse parameters."));
63     usage(argv[0]);
64     closelog();
65     exit(1);
66 }
67
68 // Floating point exceptions are masked by default.
69 _clearfp();
70 _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
71
72 // Set exception
73 __try {
74     volatile float tmp = 0;
75     switch (task) {
76     case DIVIDE_BY_ZERO:
77         writelog(_T("Ready for generate DIVIDE_BY_ZERO exception.));
78         syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
79             _T("Ready for generate DIVIDE_BY_ZERO exception.));
80         tmp = 1 / tmp;
81         writelog(_T("DIVIDE_BY_ZERO exception is generated.));
82         break;
83     case FLT_OVERFLOW:
84         // Note: floating point execution happens asynchronously.
85         // So, the exception will not be handled until the next floating
86         // point instruction.
87         writelog(_T("Ready for generate FLT_OVERFLOW exception.));
88         syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
89             _T("Ready for generate FLT_OVERFLOW exception.));
90         tmp = pow FLT_MAX, 3);
91         writelog(_T("Task: FLT_OVERFLOW exception is generated.));
92         break;
93     default:
94         break;
95     }
96 }
97 __except (EXCEPTION_EXECUTE_HANDLER) {
98     _tprintf(_T("Well, it looks like we caught something.));
99     writelog(_T("Exception is caught.));
100    syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
101        _T("Well, it looks like we caught something.));
102 }
103
104 closelog();
105 CloseHandle(eventlog);
106 exit(0);

```



```

107 }
108
109 // Usage manual
110 void usage(const _TCHAR* prog) {
111     _tprintf(_T("Usage: \n"));
112     _tprintf(_T("\t%s -d\n"), prog);
113     _tprintf(_T("\t\t\t for exception float divide by zero,\n"));
114     _tprintf(_T("\t%s -o\n"), prog);
115     _tprintf(_T("\t\t\t for exception float overflow.\n"));
116 }
117
118 void initlog(const _TCHAR* prog) {
119     _TCHAR logname[255];
120     wcsncpy_s(logname, prog);
121
122     // replace extension
123     _TCHAR* extension;
124     extension = wcsstr(logname, _T(".exe"));
125     wcsncpy_s(extension, 5, _T(".log"), 4);
126
127     // Try to open log file for append
128     if (_wfopen_s(&logfile, logname, _T("a+"))) {
129         _wprintf(_T("The following error occurred"));
130         _tprintf(_T("Can't open log file %s\n"), logname);
131         exit(1);
132     }
133
134     writelog(_T("%s is starting."), prog);
135 }
136
137 void closelog() {
138     writelog(_T("Shutting down.\n"));
139     fclose(logfile);
140 }
141
142 void writelog(_TCHAR* format, ...) {
143     _TCHAR buf[255];
144     va_list ap;
145
146     struct tm newtime;
147     __time64_t long_time;
148
149     // Get time as 64-bit integer.
150     _time64(&long_time);
151     // Convert to local time.

```

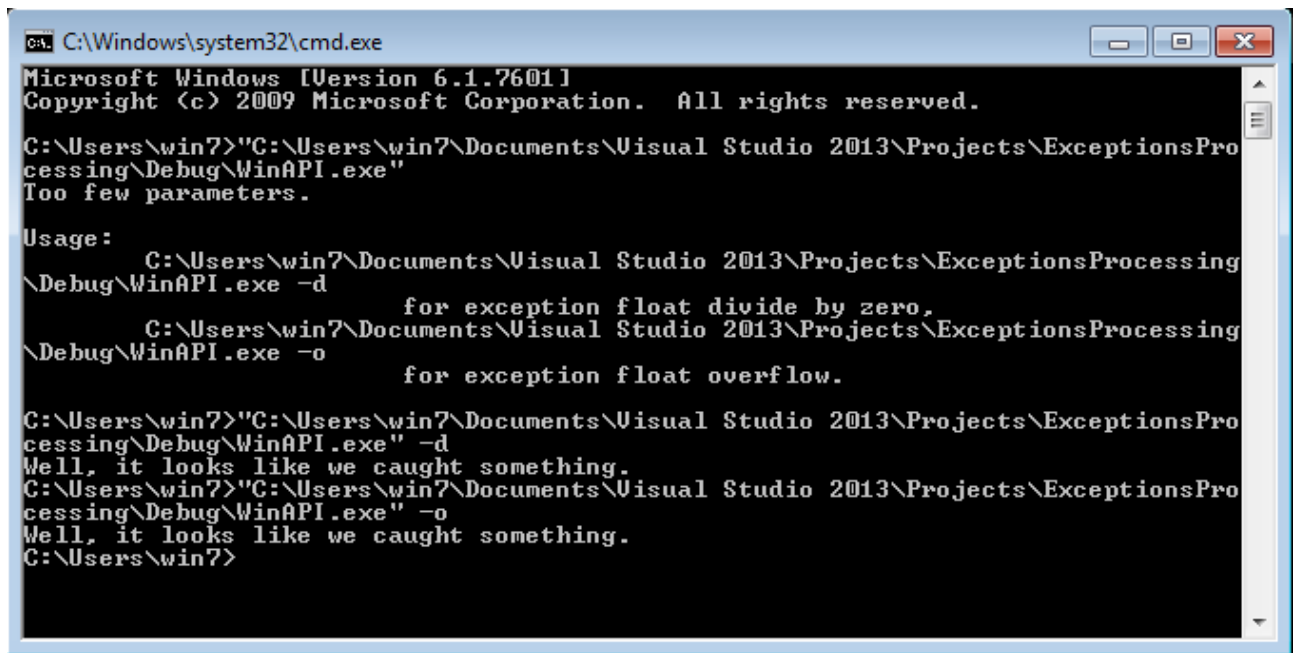
```

152 _localtime64_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     newtime.tm_min, newtime.tm_sec);
158
159 // Write date and time
160 fwprintf(logfile, _T("%s"), buf);
161 // Write all params
162 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 void syslog(WORD category, WORD identifier, LPWSTR message) {
171     LPWSTR pMessages[1] = { message };
172
173     if (!ReportEvent(
174         eventlog,          // event log handle
175         EVENTLOG_INFORMATION_TYPE, // event type
176         category,          // event category
177         identifier,         // event identifier
178         NULL,               // user security identifier
179         1,                  // number of substitution strings
180         0,                  // data size
181         (LPCWSTR*)pMessages, // pointer to strings
182         NULL)) {            // pointer to binary data buffer
183         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
184     }
185 }

```

Произведём три запуска, первый раз без аргументом (для демонстрации зависимости исключения 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.

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



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\win7>"C:\Users\win7\Documents\Visual Studio 2013\Projects\ExceptionsProcessing\Debug\WinAPI.exe"
Too few parameters.

Usage:
    C:\Users\win7\Documents\Visual Studio 2013\Projects\ExceptionsProcessing\Debug\WinAPI.exe -d
                                for exception float divide by zero.
    C:\Users\win7\Documents\Visual Studio 2013\Projects\ExceptionsProcessing\Debug\WinAPI.exe -o
                                for exception float overflow.

C:\Users\win7>"C:\Users\win7\Documents\Visual Studio 2013\Projects\ExceptionsProcessing\Debug\WinAPI.exe" -d
Well, it looks like we caught something.
C:\Users\win7>"C:\Users\win7\Documents\Visual Studio 2013\Projects\ExceptionsProcessing\Debug\WinAPI.exe" -o
Well, it looks like we caught something.
C:\Users\win7>
```

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, поэтому 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\
  ExceptionsProcessing\Debug\WinAPI.exe is starting.
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\
  ExceptionsProcessing\Debug\WinAPI.exe is starting.
6 [6/2/2015 15:33:5] Task: DIVIDE_BY_ZERO exception.
7 [6/2/2015 15:33:5] Ready for generate DIVIDE_BY_ZERO exception.
8 [6/2/2015 15:33:5] Exception is caught.
9 [6/2/2015 15:33:5] Shutting down.
10
11 [6/2/2015 15:33:10] C:\Users\win7\Documents\Visual Studio 2013\Projects\
  ExceptionsProcessing\Debug\WinAPI.exe is starting.
12 [6/2/2015 15:33:10] Task: FLT_OVERFLOW exception.
13 [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.
```

Из примечательного в этом коде ещё строка 173. В ней показан вызов команды ReportEvent[3], которая непосредственно передаёт строку в системный журнал. Синтаксис команды следующий:

```
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 информации, 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-

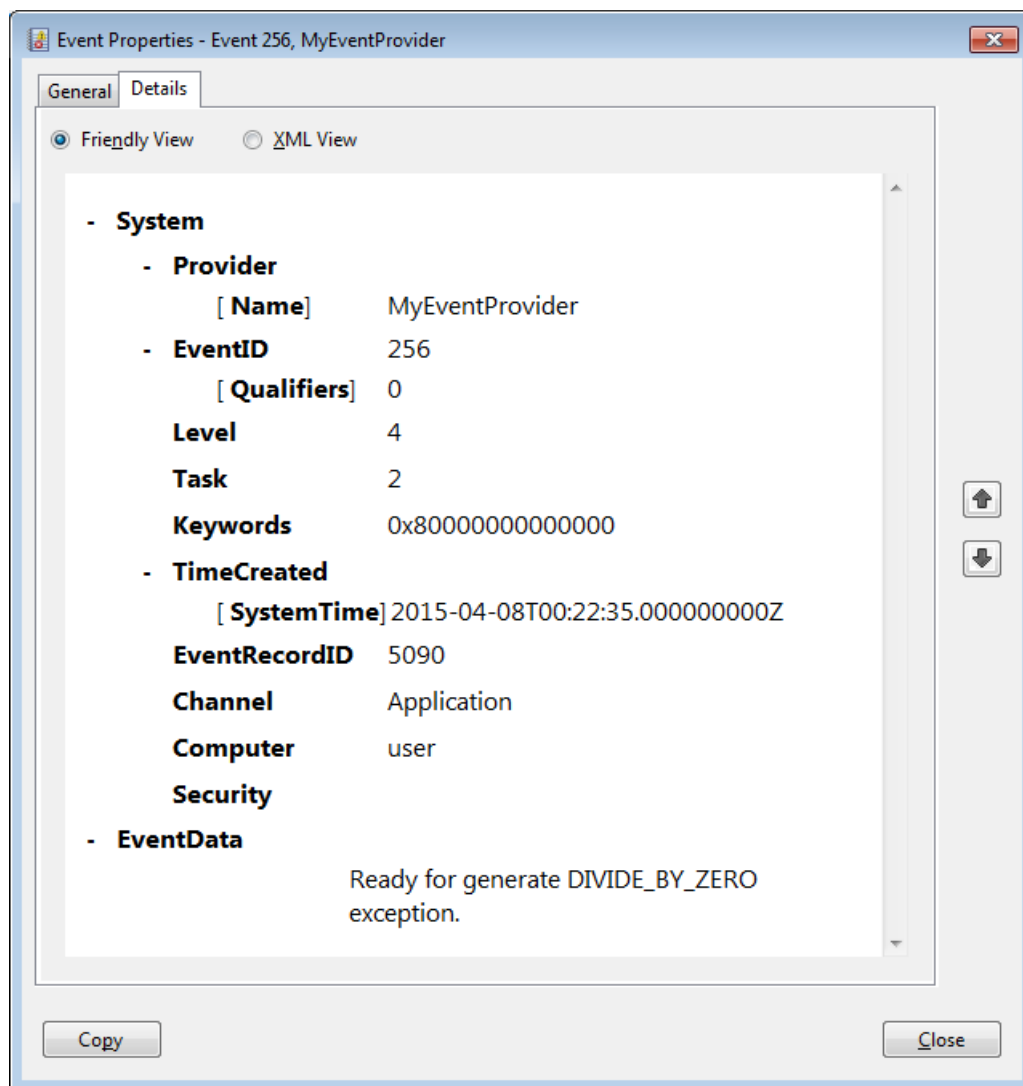


Рис. 8: Просмотр события в "дружелюбном" виде (альтернативой которому является XML).

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

В отличие от OllyDbg, WinDbg при первом запуске имеет достаточно неприятный интерфейс 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 C, так и в ассемблерном коде), а под ним диалоговое окно, в котором отображается результат выполнения запросов пользователя. Центральное окно представляет таблицу значений регистров цен-

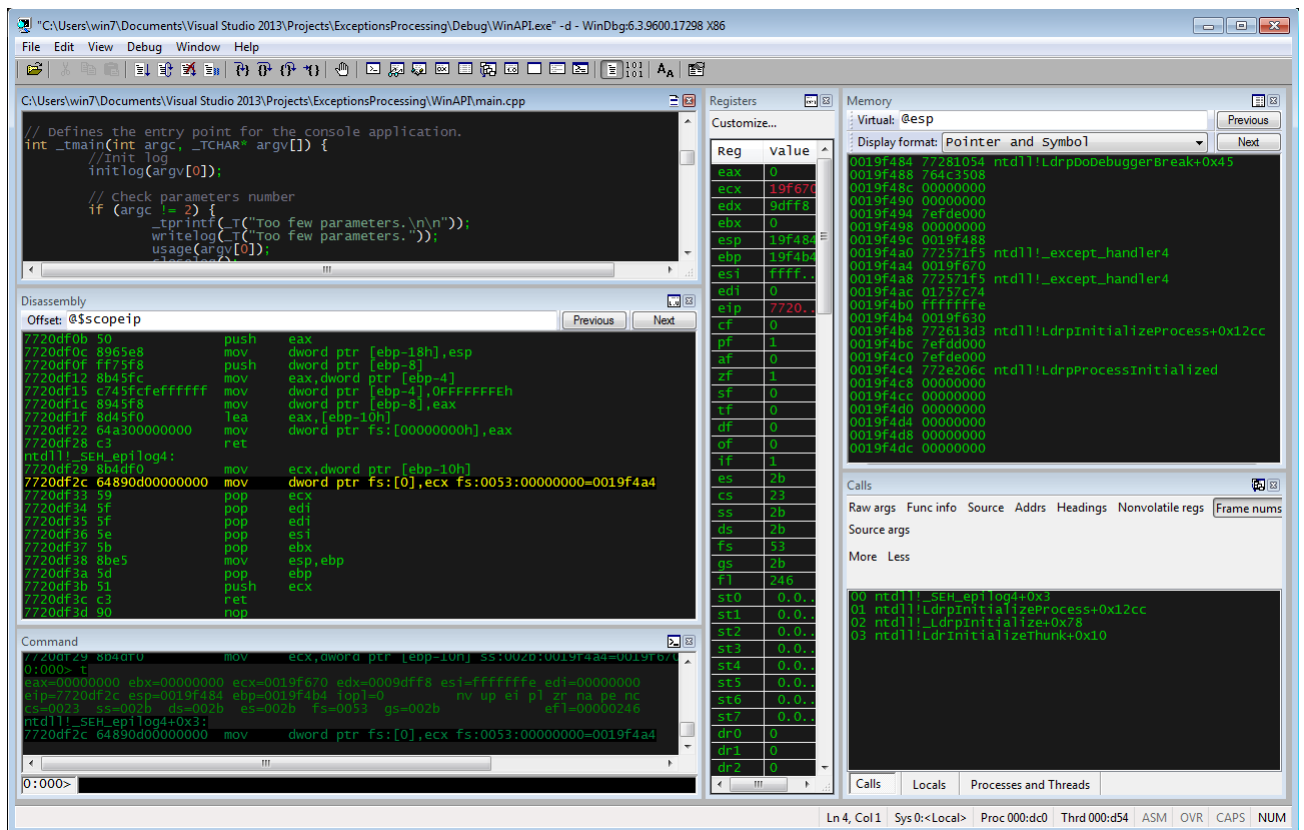


Рис. 9: Запуск WinAPI.exe под отладчиком WinDbg.

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.
- pcr – шаг к следующему вызову или возврату.

Точки останова:

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

- bl – список точек останова.
- bd – <число> убрать точку останова 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.

Благодаря 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 сгенерировано в процессе работы программы (смотри листинг 5)[1]. В первом случае (строка 86) она участвует в сравнении с макро-константой `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  /* Task 2.
2     Get the exceptions code using the GetExceptionCode gunction:
3     - Use this function in the filter expression;
4     - Use this function in the handler.
5  */
6
7  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!
8  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
9  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)
10
11 #include <stdio.h>
12 #include <tchar.h>
13 #include <cstring>
14 #include <cfloat>
15 #include <cmath>
16 #include <excpt.h>
17 #include <windows.h>
18 #include <time.h>
19
20 #include "messages.h"
21
22 // log
23 FILE* logfile;
24 HANDLE eventlog;
25
```



```

26 void usage(const _TCHAR* prog);
27 void initlog(const _TCHAR* prog);
28 void closelog();
29 void writelog(_TCHAR* format, ...);
30 void syslog(WORD category, WORD identifier, LPWSTR message);
31
32 // Task switcher
33 enum {
34     DIVIDE_BY_ZERO,
35     FLT_OVERFLOW
36 } task;
37
38 // Defines the entry point for the console application.
39 int _tmain(int argc, _TCHAR* argv[]) {
40     //Init log
41     initlog(argv[0]);
42     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
43
44     // Check parameters number
45     if (argc != 2) {
46         _tprintf(_T("Too few parameters.\n\n"));
47         writelog(_T("Too few parameters."));
48         usage(argv[0]);
49         closelog();
50         exit(1);
51     }
52
53     // Set task
54     if (!_tcscmp(_T("-d"), argv[1])) {
55         task = DIVIDE_BY_ZERO;
56         writelog(_T("Task: DIVIDE_BY_ZERO exception."));
57     }
58     else if (!_tcscmp(_T("-o"), argv[1])) {
59         task = FLT_OVERFLOW;
60         writelog(_T("Task: FLT_OVERFLOW exception."));
61     }
62     else {
63         _tprintf(_T("Can't parse parameters.\n\n"));
64         writelog(_T("Can't parse parameters."));
65         usage(argv[0]);
66         closelog();
67         exit(1);
68     }
69
70     // Floating point exceptions are masked by default.

```

```

71 _clearfp();
72 _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
73
74 // Set exception
75 volatile float tmp = 0;
76 switch (task) {
77 case DIVIDE_BY_ZERO:
78     __try {
79         writelog(_T("Ready for generate DIVIDE_BY_ZERO exception.));
80         syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
81             _T("Ready for generate DIVIDE_BY_ZERO exception.));
82         tmp = 1 / tmp;
83         writelog(_T("DIVIDE_BY_ZERO exception is generated.));
84     }
85     // Use GetExceptionCode() function in the filter expression;
86     __except ((GetExceptionCode() == EXCEPTION_FLT_DIVIDE_BY_ZERO) ?
87         EXCEPTION_EXECUTE_HANDLER :
88             EXCEPTION_CONTINUE_SEARCH)
89     {
90         _tprintf(_T("Caught exception is: EXCEPTION_FLT_DIVIDE_BY_ZERO"));
91         writelog(_T("Caught exception is: EXCEPTION_FLT_DIVIDE_BY_ZERO"));
92         syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
93             _T("Caught exception is: EXCEPTION_FLT_DIVIDE_BY_ZERO.));
94     }
95     break;
96 case FLT_OVERFLOW:
97     __try {
98         // Note: floating point execution happens asynchronously.
99         // So, the exception will not be handled until the next
100         // floating point instruction.
101         writelog(_T("Ready for generate FLT_OVERFLOW exception.));
102         syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
103             _T("Ready for generate FLT_OVERFLOW exception.));
104         tmp = pow FLT_MAX, 3);
105         writelog(_T("Task: FLT_OVERFLOW exception is generated.));
106     }
107     // Use GetExceptionCode() function in the handler.
108     __except (EXCEPTION_EXECUTE_HANDLER) {
109         if (GetExceptionCode() == EXCEPTION_FLT_OVERFLOW) {
110             writelog(_T("Caught exception is: EXCEPTION_FLT_OVERFLOW"));
111             _tprintf(_T("Caught exception is: EXCEPTION_FLT_OVERFLOW"));
112             syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
113                 _T("Caught exception is: EXCEPTION_FLT_OVERFLOW.));
114         }
115         else {

```

```

116         writelog(_T("UNKNOWN exception: %x\n"), GetExceptionCode());
117         _tprintf(_T("UNKNOWN exception: %x\n"), GetExceptionCode());
118     }
119 }
120     break;
121 default:
122     break;
123 }
124     closelog();
125     CloseHandle(eventlog);
126     exit(0);
127 }
128
129 // Usage manual
130 void 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 void initlog(const _TCHAR* prog) {
139     _TCHAR logname[255];
140     wcsncpy_s(logname, prog);
141
142     // replace extension
143     _TCHAR* extension;
144     extension = wcsstr(logname, _T(".exe"));
145     wcsncpy_s(extension, 5, _T(".log"), 4);
146
147     // Try to open log file for append
148     if (_wfopen_s(&logfile, logname, _T("a+"))) {
149         _wprintf(_T("The following error occurred"));
150         _tprintf(_T("Can't open log file %s\n"), logname);
151         exit(1);
152     }
153
154     writelog(_T("%s is starting."), prog);
155 }
156
157 void closelog() {
158     writelog(_T("Shutting down.\n"));
159     fclose(logfile);
160 }

```

```

161
162 void writelog(_TCHAR* format, ...) {
163     _TCHAR buf[255];
164     va_list ap;
165
166     struct tm newtime;
167     __time64_t long_time;
168
169     // Get time as 64-bit integer.
170     _time64(&long_time);
171     // Convert to local time.
172     _localtime64_s(&newtime, &long_time);
173
174     // Convert to normal representation.
175     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
176         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
177         newtime.tm_min, newtime.tm_sec);
178
179     // Write date and time
180     fwprintf(logfile, _T("%s"), buf);
181     // Write all params
182     va_start(ap, format);
183     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
184     fwprintf(logfile, _T("%s"), buf);
185     va_end(ap);
186     // New sting
187     fwprintf(logfile, _T("\n"));
188 }
189
190 void syslog(WORD category, WORD identifier, LPWSTR message) {
191     LPWSTR pMessages[1] = { message };
192
193     if (!ReportEvent(
194         eventlog,           // event log handle
195         EVENTLOG_INFORMATION_TYPE, // event type
196         category,          // event category
197         identifier,         // event identifier
198         NULL,               // user security identifier
199         1,                  // number of substitution strings
200         0,                  // data size
201         (LPCWSTR*)pMessages, // pointer to strings
202         NULL)) {            // pointer to binary data buffer
203         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
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\
  ExceptionsProcessing\Debug\GetExceptionCode.exe is starting.
2 [6/2/2015 16:42:36] Task: DIVIDE_BY_ZERO exception.
3 [6/2/2015 16:42:36] Ready for generate DIVIDE_BY_ZERO exception.
4 [6/2/2015 16:42:36] Caught exception is: EXCEPTION_FLT_DIVIDE_BY_ZERO
5 [6/2/2015 16:42:36] Shutting down.
6
7 [6/2/2015 16:42:39] C:\Users\win7\Documents\Visual Studio 2013\Projects\
  ExceptionsProcessing\Debug\GetExceptionCode.exe is starting.
8 [6/2/2015 16:42:39] Task: FLT_OVERFLOW exception.
9 [6/2/2015 16:42:39] Ready for generate FLT_OVERFLOW exception.
10 [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 [1].

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

```
1  /* Task 3.
2  Create your own filter function.
3  */
4
5  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!
6  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
7  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)
8
9  #include <stdio.h>
10 #include <tchar.h>
11 #include <cstring>
12 #include <cfloat>
13 #include <cmath>
14 #include <excpt.h>
15 #include <windows.h>
16 #include <time.h>
17
18 #include "messages.h"
19
20 // log
21 FILE* logfile;
22 HANDLE eventlog;
23
24 void usage(const _TCHAR* prog);
25 void initlog(const _TCHAR* prog);
26 void closelog();
27 void writelog(_TCHAR* format, ...);
28 LONG Filter(DWORD dwExceptionCode);
```

```

29 void syslog(WORD category, WORD identifier, LPWSTR message);
30
31 // Task switcher
32 enum {
33     DIVIDE_BY_ZERO,
34     FLT_OVERFLOW
35 } task;
36
37 // Defines the entry point for the console application.
38 int _tmain(int argc, _TCHAR* argv[]) {
39     //Init log
40     initlog(argv[0]);
41     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
42
43     // Check parameters number
44     if (argc != 2) {
45         _tprintf(_T("Too few parameters.\n\n"));
46         writelog(_T("Too few parameters."));
47         usage(argv[0]);
48         closelog();
49         exit(1);
50     }
51
52     // Set task
53     if (!_tcscmp(_T("-d"), argv[1])) {
54         task = DIVIDE_BY_ZERO;
55         writelog(_T("Task: DIVIDE_BY_ZERO exception."));
56     }
57     else if (!_tcscmp(_T("-o"), argv[1])) {
58         task = FLT_OVERFLOW;
59         writelog(_T("Task: FLT_OVERFLOW exception."));
60     }
61     else {
62         _tprintf(_T("Can't parse parameters.\n\n"));
63         writelog(_T("Can't parse parameters."));
64         usage(argv[0]);
65         closelog();
66         exit(1);
67     }
68
69     // Floating point exceptions are masked by default.
70     _clearfp();
71     _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
72
73     // Set exception

```

```

74  __try {
75      volatile float tmp = 0;
76      switch (task) {
77      case DIVIDE_BY_ZERO:
78          writelog(_T("Ready for generate DIVIDE_BY_ZERO exception.));
79          syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
80              _T("Ready for generate DIVIDE_BY_ZERO exception.));
81          tmp = 1 / tmp;
82          writelog(_T("DIVIDE_BY_ZERO exception is generated.));
83          break;
84      case FLT_OVERFLOW:
85          // Note: floating point execution happens asynchronously.
86          // So, the exception will not be handled until the next floating
87          // point instruction.
88          writelog(_T("Ready for generate FLT_OVERFLOW exception.));
89          syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
90              _T("Ready for generate FLT_OVERFLOW exception.));
91          tmp = pow(FLT_MAX, 3);
92          writelog(_T("Task: FLT_OVERFLOW exception is generated.));
93          break;
94      default:
95          break;
96      }
97  }
98  // Own filter function.
99  __except (Filter(GetExceptionCode())) {
100      printf("Caught exception is: ");
101      switch (GetExceptionCode()){
102      case EXCEPTION_FLT_DIVIDE_BY_ZERO:
103          _tprintf(_T("Caught exception is: EXCEPTION_FLT_DIVIDE_BY_ZERO"));
104          writelog(_T("Caught exception is: EXCEPTION_FLT_DIVIDE_BY_ZERO"));
105          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"), GetExceptionCode());
116          writelog(_T("UNKNOWN exception: %x\n"), GetExceptionCode());
117      }
118  }

```



```

119     closelog();
120     CloseHandle(eventlog);
121     exit(0);
122 }
123
124 // Own filter function.
125 LONG Filter(DWORD dwExceptionCode) {
126     _tprintf(_T("Filter function used"));
127     if (dwExceptionCode == EXCEPTION_FLT_DIVIDE_BY_ZERO || dwExceptionCode ==
        EXCEPTION_FLT_OVERFLOW)
128         return EXCEPTION_EXECUTE_HANDLER;
129     return EXCEPTION_CONTINUE_SEARCH;
130 }
131
132 // Usage manual
133 void usage(const _TCHAR* prog) {
134     _tprintf(_T("Usage: \n"));
135     _tprintf(_T("\t%s -d\n"), prog);
136     _tprintf(_T("\t\t\t for exception float divide by zero,\n"));
137     _tprintf(_T("\t%s -o\n"), prog);
138     _tprintf(_T("\t\t\t for exception float overflow.\n"));
139 }
140
141 void initlog(const _TCHAR* prog) {
142     _TCHAR logname[255];
143     wcscpy_s(logname, prog);
144
145     // replace extension
146     _TCHAR* extension;
147     extension = wcsstr(logname, _T(".exe"));
148     wcsncpy_s(extension, 5, _T(".log"), 4);
149
150     // Try to open log file for append
151     if (_wfopen_s(&logfile, logname, _T("a+"))) {
152         _wprintf(_T("The following error occurred"));
153         _tprintf(_T("Can't open log file %s\n"), logname);
154         exit(1);
155     }
156
157     writelog(_T("%s is starting."), prog);
158 }
159
160 void closelog() {
161     writelog(_T("Shutting down.\n"));
162     fclose(logfile);

```

```

163 }
164
165 void writelog(_TCHAR* format, ...) {
166     _TCHAR buf[255];
167     va_list ap;
168
169     struct tm newtime;
170     __time64_t long_time;
171
172     // Get time as 64-bit integer.
173     _time64(&long_time);
174     // Convert to local time.
175     _localtime64_s(&newtime, &long_time);
176
177     // Convert to normal representation.
178     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
179         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
180         newtime.tm_min, newtime.tm_sec);
181
182     // Write date and time
183     fwprintf(logfile, _T("%s"), buf);
184     // Write all params
185     va_start(ap, format);
186     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
187     fwprintf(logfile, _T("%s"), buf);
188     va_end(ap);
189     // New sting
190     fwprintf(logfile, _T("\n"));
191 }
192
193 void syslog(WORD category, WORD identifier, LPWSTR message) {
194     LPWSTR pMessages[1] = { message };
195
196     if (!ReportEvent(
197         eventlog,           // event log handle
198         EVENTLOG_INFORMATION_TYPE, // event type
199         category,           // event category
200         identifier,         // event identifier
201         NULL,               // user security identifier
202         1,                  // number of substitution strings
203         0,                  // data size
204         (LPCWSTR*)pMessages, // pointer to strings
205         NULL)) {            // pointer to binary data buffer
206         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
207     }

```

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

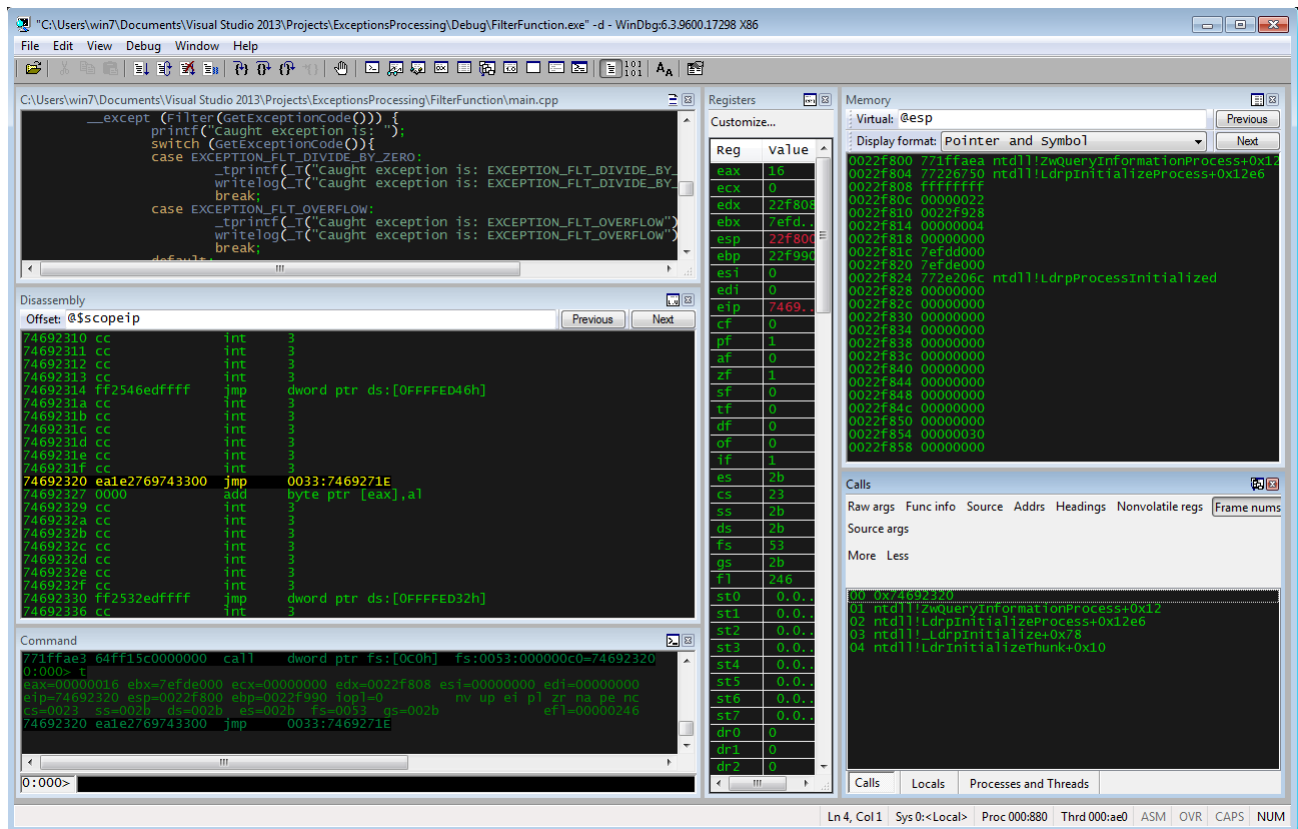


Рис. 10: Передача управления обработчику, после отработки функции-фильтра

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\
  ExceptionsProcessing\Debug\FilterFunction.exe is starting.
2 [6/2/2015 16:45:29] Task: DIVIDE_BY_ZERO exception.
3 [6/2/2015 16:45:29] Ready for generate DIVIDE_BY_ZERO exception.
4 [6/2/2015 16:45:29] Caught exception is: EXCEPTION_FLT_DIVIDE_BY_ZERO
5 [6/2/2015 16:45:29] Shutting down.
6
7 [6/2/2015 16:45:34] C:\Users\win7\Documents\Visual Studio 2013\Projects\
  ExceptionsProcessing\Debug\FilterFunction.exe is starting.

```

```
8 [6/2/2015 16:45:34] Task: FLT_OVERFLOW exception.  
9 [6/2/2015 16:45:34] Ready for generate FLT_OVERFLOW exception.  
10 [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 логических операций, но и искусственным образом, вызвав функцию RaiseException. Она обладает 4-я параметрами, но наиболее важным является первый, который определяет тип возбуждаемого исключения[1]. Работа этой функции показана в листинге 9.

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

```
1  /* Task 4.
2  Get information about the exception using the GetExceptionInformation()
   fnc;
3  throw an exception using the RaiseException() function.
4  */
5
6  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!
7  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
8  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)
9
10 #include <stdio.h>
11 #include <tchar.h>
12 #include <cstring>
13 #include <cfloat>
14 #include <cmath>
15 #include <excpt.h>
16 #include <windows.h>
17 #include <time.h>
18
19 #include "messages.h"
20
21 // log
22 FILE* logfile;
23 HANDLE eventlog;
24
25 void usage(const _TCHAR* prog);
26 void initlog(const _TCHAR* prog);
```

```

27 void closelog();
28 void writelog(_TCHAR* format, ...);
29 LONG Filter(DWORD dwExceptionCode, const _EXCEPTION_POINTERS *ep);
30 void syslog(WORD category, WORD identifier, LPWSTR message);
31
32 // Task switcher
33 enum {
34     DIVIDE_BY_ZERO,
35     FLT_OVERFLOW
36 } task;
37
38 // Defines the entry point for the console application.
39 int _tmain(int argc, _TCHAR* argv[]) {
40     //Init log
41     initlog(argv[0]);
42     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
43
44     // Check parameters number
45     if (argc != 2) {
46         _tprintf(_T("Too few parameters.\n\n"));
47         writelog(_T("Too few parameters."));
48         usage(argv[0]);
49         closelog();
50         exit(1);
51     }
52
53     // Set task
54     if (!_tcscmp(_T("-d"), argv[1])) {
55         task = DIVIDE_BY_ZERO;
56         writelog(_T("Task: DIVIDE_BY_ZERO exception."));
57     }
58     else if (!_tcscmp(_T("-o"), argv[1])) {
59         task = FLT_OVERFLOW;
60         writelog(_T("Task: FLT_OVERFLOW exception."));
61     }
62     else {
63         _tprintf(_T("Can't parse parameters.\n\n"));
64         writelog(_T("Can't parse parameters."));
65         usage(argv[0]);
66         closelog();
67         exit(1);
68     }
69
70     // Floating point exceptions are masked by default.
71     _clearfp();

```

```

72 _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
73
74 __try {
75     switch (task) {
76     case DIVIDE_BY_ZERO:
77         // throw an exception using the RaiseException() function
78         writelog(_T("Ready for generate DIVIDE_BY_ZERO exception.));
79         syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
80             _T("Ready for generate DIVIDE_BY_ZERO exception.));
81         RaiseException(EXCEPTION_FLT_DIVIDE_BY_ZERO,
82             EXCEPTION_NONCONTINUABLE, 0, NULL);
83         writelog(_T("DIVIDE_BY_ZERO exception is generated.));
84         break;
85     case FLT_OVERFLOW:
86         // throw an exception using the RaiseException() function
87         writelog(_T("Ready for generate FLT_OVERFLOW exception.));
88         syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
89             _T("Ready for generate FLT_OVERFLOW exception.));
90         RaiseException(EXCEPTION_FLT_OVERFLOW,
91             EXCEPTION_NONCONTINUABLE, 0, NULL);
92         writelog(_T("Task: FLT_OVERFLOW exception is generated.));
93         break;
94     default:
95         break;
96     }
97 }
98 __except (Filter(GetExceptionCode(), GetExceptionInformation())) {
99     // There is nothing to do, everything is done in the filter function.
100 }
101 closelog();
102 CloseHandle(eventlog);
103 exit(0);
104 }
105
106 LONG Filter(DWORD dwExceptionCode, const _EXCEPTION_POINTERS *
    ExceptionPointers) {
107     enum { size = 200 };
108     _TCHAR buf[size] = { '\0' };
109     const _TCHAR* err = _T("Fatal error!\nexeption code: 0x");
110     const _TCHAR* mes = _T("\nProgram terminate!");
111     if (ExceptionPointers)
112         // Get information about the exception using the GetExceptionInformation
113         swprintf_s(buf, _T("%s%x%s%x%s%x"), err,
114             ExceptionPointers->ExceptionRecord->ExceptionCode,
115             _T(", data adress: 0x"),

```

```

116     ExceptionPointers->ExceptionRecord->ExceptionInformation[1],
117     _T(", instruction address: 0x"),
118     ExceptionPointers->ExceptionRecord->ExceptionAddress, mes);
119 else
120     swprintf_s(buf, _T("%s%x%s"), err, dwExceptionCode, 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 void usage(const _TCHAR* prog) {
131     _tprintf(_T("Usage: \n"));
132     _tprintf(_T("\t%s -d\n"), prog);
133     _tprintf(_T("\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\t for exception float overflow.\n"));
136 }
137
138 void initlog(const _TCHAR* prog) {
139     _TCHAR logname[255];
140     wcsncpy_s(logname, prog);
141
142     // replace extension
143     _TCHAR* extension;
144     extension = wcsstr(logname, _T(".exe"));
145     wcsncpy_s(extension, 5, _T(".log"), 4);
146
147     // Try to open log file for append
148     if (_wfopen_s(&logfile, logname, _T("a+"))) {
149         _wprintf(_T("The following error occurred"));
150         _tprintf(_T("Can't open log file %s\n"), logname);
151         exit(1);
152     }
153
154     writelog(_T("%s is starting."), prog);
155 }
156
157 void closelog() {
158     writelog(_T("Shutting down.\n"));
159     fclose(logfile);
160 }

```



```

161
162 void writelog(_TCHAR* format, ...) {
163     _TCHAR buf[255];
164     va_list ap;
165
166     struct tm newtime;
167     __time64_t long_time;
168
169     // Get time as 64-bit integer.
170     _time64(&long_time);
171     // Convert to local time.
172     _localtime64_s(&newtime, &long_time);
173
174     // Convert to normal representation.
175     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
176         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
177         newtime.tm_min, newtime.tm_sec);
178
179     // Write date and time
180     fwprintf(logfile, _T("%s"), buf);
181     // Write all params
182     va_start(ap, format);
183     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
184     fwprintf(logfile, _T("%s"), buf);
185     va_end(ap);
186     // New sting
187     fwprintf(logfile, _T("\n"));
188 }
189
190 void syslog(WORD category, WORD identifier, LPWSTR message) {
191     LPWSTR pMessages[1] = { message };
192
193     if (!ReportEvent(
194         eventlog,           // event log handle
195         EVENTLOG_INFORMATION_TYPE, // event type
196         category,          // event category
197         identifier,         // event identifier
198         NULL,              // user security identifier
199         1,                 // number of substitution strings
200         0,                 // data size
201         (LPCWSTR*)pMessages, // pointer to strings
202         NULL)) {           // pointer to binary data buffer
203         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
204     }
205 }

```

Информацию об исключении можно получить из функции `GetExceptionInformation`, которая, 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 `GetExceptionInformation` является то, что ее можно вызывать 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 исключении. Обработчик исключения находится выше по стеку, и когда ему будет возвращено управление от функции фильтра стек уже будет зачищен.

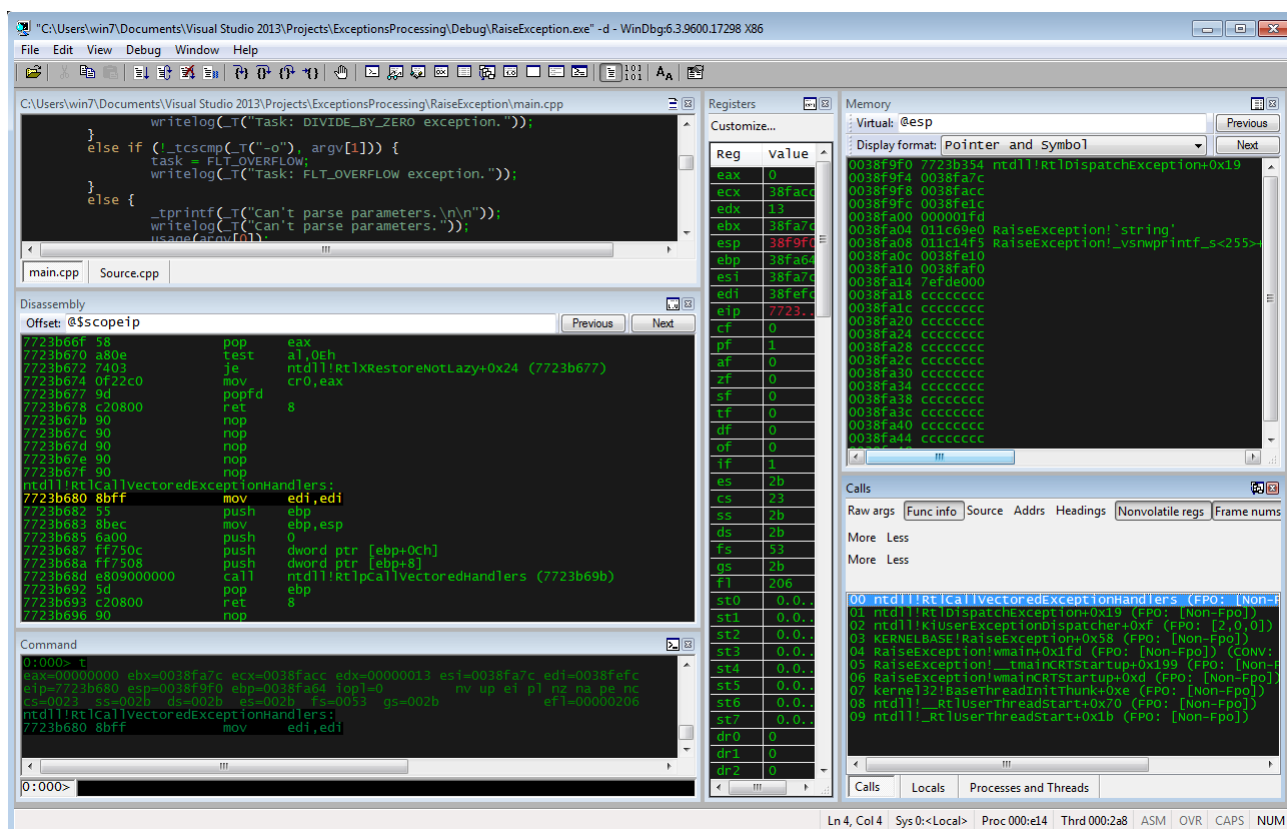
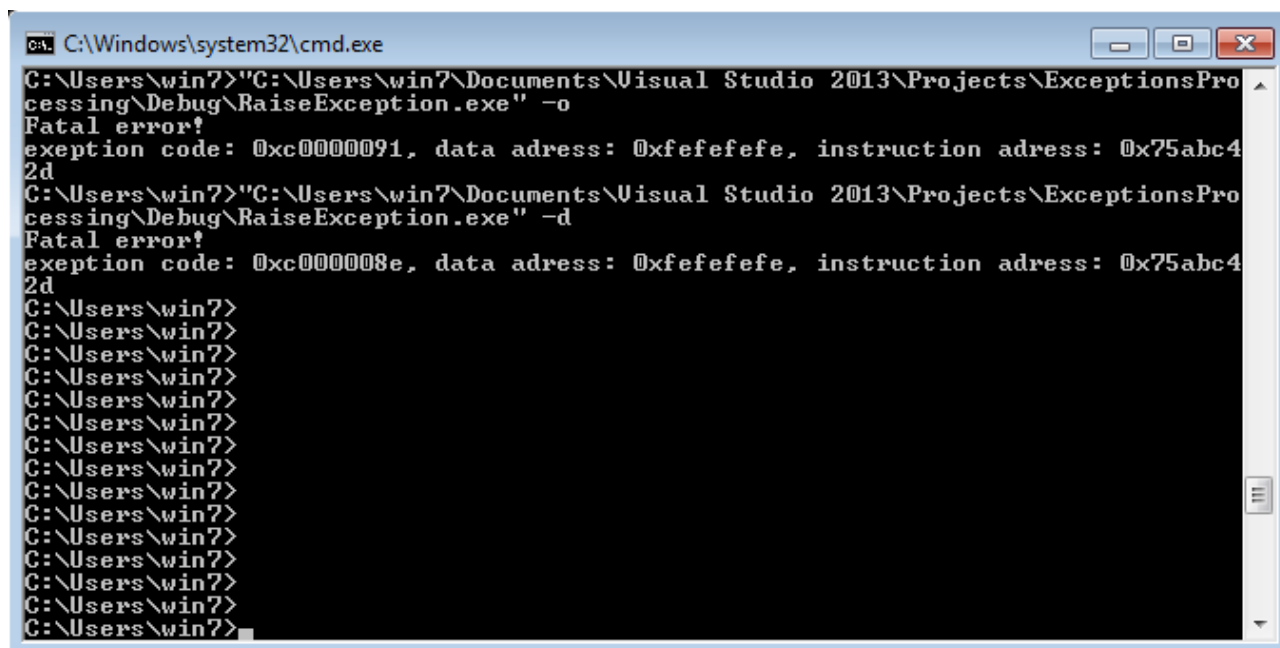


Рис. 11: Информация об исключении доступна в процессе работы функции-фильтра



```
C:\Windows\system32\cmd.exe
C:\Users\win7>"C:\Users\win7\Documents\Visual Studio 2013\Projects\ExceptionsProcessing\Debug\RaiseException.exe" -o
Fatal error!
exception code: 0xc0000091, data address: 0xfefefefe, instruction address: 0x75abc42d
C:\Users\win7>"C:\Users\win7\Documents\Visual Studio 2013\Projects\ExceptionsProcessing\Debug\RaiseException.exe" -d
Fatal error!
exception code: 0xc000008e, data address: 0xfefefefe, instruction address: 0x75abc42d
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
C:\Users\win7>
```

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\
  ExceptionsProcessing\Debug\RaiseException.exe is starting.
2 [6/2/2015 17:7:22] Task: FLT_OVERFLOW exception.
3 [6/2/2015 17:7:22] Ready for generate FLT_OVERFLOW exception.
4 [6/2/2015 17:7:22] Fatal error!
5 exception code: 0xc0000091, data address: 0xfefefefe, instruction address: 0
  x75abc42d
6 [6/2/2015 17:7:22] Shutting down.
7
8 [6/2/2015 17:7:26] C:\Users\win7\Documents\Visual Studio 2013\Projects\
  ExceptionsProcessing\Debug\RaiseException.exe is starting.
9 [6/2/2015 17:7:26] Task: DIVIDE_BY_ZERO exception.
10 [6/2/2015 17:7:26] Ready for generate DIVIDE_BY_ZERO exception.
11 [6/2/2015 17:7:26] Fatal error!
12 exception code: 0xc000008e, data address: 0xfefefefe, instruction address: 0
  x75abc42d
13 [6/2/2015 17:7:26] Shutting down.
```

Необрабатываемые исключения

Если ни один из установленных программистом обработчиков не подошла для обработки исключения (либо программист вообще не установил ни один обработчик), то вызывается функция `UnhandledExceptionFilter`, которая выполняет проверку, запущен ли процесс под отладчиком, и информирует процесс, если отладчик доступен[1]. Далее, функция 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 определяется в строках 119 и 120.

Листинг 11: Необработанные исключения

(src/ExceptionsProcessing/UnhandleExceptionFilter/main.cpp)

```
1  /* Task 5.
2  Use the UnhandledExceptionFilter and SetUnhandledeceptionfilter
3  for unhandled exceptions;.
4  */
5
6  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!
7  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
8  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)
9
10 #include <stdio.h>
11 #include <tchar.h>
12 #include <cstring>
13 #include <cfloat>
14 #include <cmath>
15 #include <except.h>
16 #include <windows.h>
17 #include <time.h>
18
19 #include "messages.h"
```

```

20
21 // log
22 FILE* logfile;
23 HANDLE eventlog;
24
25 void usage(const _TCHAR* prog);
26 void initlog(const _TCHAR* prog);
27 void closelog();
28 void writelog(_TCHAR* format, ...);
29 void syslog(WORD category, WORD identifier, LPWSTR message);
30
31 // prototype
32 LONG WINAPI MyUnhandledExceptionFilter(EXCEPTION_POINTERS* ExceptionInfo);
33
34 // Task switcher
35 enum {
36     DIVIDE_BY_ZERO,
37     FLT_OVERFLOW
38 } task;
39
40 // Defines the entry point for the console application.
41 int _tmain(int argc, _TCHAR* argv[]) {
42     //Init log
43     initlog(argv[0]);
44     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
45
46     // Check parameters number
47     if (argc != 2) {
48         _tprintf(_T("Too few parameters.\n\n"));
49         writelog(_T("Too few parameters. "));
50         usage(argv[0]);
51         closelog();
52         exit(1);
53     }
54
55     // Set task
56     if (!_tcscmp(_T("-d"), argv[1])) {
57         task = DIVIDE_BY_ZERO;
58         writelog(_T("Task: DIVIDE_BY_ZERO exception. "));
59     }
60     else if (!_tcscmp(_T("-o"), argv[1])) {
61         task = FLT_OVERFLOW;
62         writelog(_T("Task: FLT_OVERFLOW exception. "));
63     }
64     else {

```

```

65     _tprintf(_T("Can't parse parameters.\n\n"));
66     writelog(_T("Can't parse parameters."));
67     usage(argv[0]);
68     closelog();
69     exit(1);
70 }
71
72 // Floating point exceptions are masked by default.
73 _clearfp();
74 _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
75
76 volatile float tmp = 0;
77 ::SetUnhandledExceptionFilter(MyUnhandledExceptionFilter);
78
79 switch (task) {
80 case DIVIDE_BY_ZERO:
81     // throw an exception using the RaiseException() function
82     writelog(_T("Ready for generate DIVIDE_BY_ZERO exception.));
83     syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
84         _T("Ready for generate DIVIDE_BY_ZERO exception.));
85     RaiseException(EXCEPTION_FLT_DIVIDE_BY_ZERO,
86         EXCEPTION_EXECUTE_FAULT, 0, NULL);
87     writelog(_T("DIVIDE_BY_ZERO exception is generated.));
88     break;
89 case FLT_OVERFLOW:
90     // throw an exception using the RaiseException() function
91     writelog(_T("Ready for generate FLT_OVERFLOW exception.));
92     syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
93         _T("Ready for generate FLT_OVERFLOW exception.));
94     RaiseException(EXCEPTION_FLT_OVERFLOW,
95         EXCEPTION_EXECUTE_FAULT, 0, NULL);
96     writelog(_T("Task: FLT_OVERFLOW exception is generated.));
97     break;
98 default:
99     break;
100 }
101
102 closelog();
103 CloseHandle(eventlog);
104 exit(0);
105 }
106
107 LONG WINAPI MyUnhandledExceptionFilter(EXCEPTION_POINTERS* ExceptionInfo) {
108     enum { size = 200 };
109     _TCHAR buf[size] = { '\0' };

```

```

110  const _TCHAR* err = _T("Unhandled exception!\nexception code : 0x");
111  // Get information about the exception using the GetExceptionInformation
112  swprintf_s(buf, _T("%s%x%s%x%s%x"), err, ExceptionInfo->ExceptionRecord->
    ExceptionCode,
113    _T(", data address: 0x"), ExceptionInfo->ExceptionRecord->
    ExceptionInformation[1],
114    _T(", instruction address: 0x"), ExceptionInfo->ExceptionRecord->
    ExceptionAddress);
115  _tprintf(_T("%s"), buf);
116  writelog(_T("%s"), buf);
117  syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION, buf);
118
119  return EXCEPTION_CONTINUE_SEARCH;
120  //return EXCEPTION_EXECUTE_HANDLER;
121 }
122
123 // Usage manual
124 void usage(const _TCHAR* prog) {
125     _tprintf(_T("Usage: \n"));
126     _tprintf(_T("\t%s -d\n"), prog);
127     _tprintf(_T("\t\t\t for exception float divide by zero,\n"));
128     _tprintf(_T("\t%s -o\n"), prog);
129     _tprintf(_T("\t\t\t for exception float overflow.\n"));
130 }
131
132 void initlog(const _TCHAR* prog) {
133     _TCHAR logname[255];
134     wcsncpy_s(logname, prog);
135
136     // replace extension
137     _TCHAR* extension;
138     extension = wcsstr(logname, _T(".exe"));
139     wcsncpy_s(extension, 5, _T(".log"), 4);
140
141     // Try to open log file for append
142     if (_wfopen_s(&logfile, logname, _T("a+"))) {
143         _wpererror(_T("The following error occurred"));
144         _tprintf(_T("Can't open log file %s\n"), logname);
145         exit(1);
146     }
147
148     writelog(_T("%s is starting."), prog);
149 }
150
151 void closelog() {

```

```

152     writelog(_T("Shutting down.\n"));
153     fclose(logfile);
154 }
155
156 void writelog(_TCHAR* format, ...) {
157     _TCHAR buf[255];
158     va_list ap;
159
160     struct tm newtime;
161     __time64_t long_time;
162
163     // Get time as 64-bit integer.
164     _time64(&long_time);
165     // Convert to local time.
166     _localtime64_s(&newtime, &long_time);
167
168     // Convert to normal representation.
169     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
170         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
171         newtime.tm_min, newtime.tm_sec);
172
173     // Write date and time
174     fwprintf(logfile, _T("%s"), buf);
175     // Write all params
176     va_start(ap, format);
177     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
178     fwprintf(logfile, _T("%s"), buf);
179     va_end(ap);
180     // New sting
181     fwprintf(logfile, _T("\n"));
182     fflush(logfile);
183 }
184
185 void syslog(WORD category, WORD identifier, LPWSTR message) {
186     LPWSTR pMessages[1] = { message };
187
188     if (!ReportEvent(
189         eventlog,           // event log handle
190         EVENTLOG_INFORMATION_TYPE, // event type
191         category,           // event category
192         identifier,         // event identifier
193         NULL,               // user security identifier
194         1,                  // number of substitution strings
195         0,                  // data size
196         (LPCWSTR*)pMessages, // pointer to strings

```



```

197     NULL)) {                // pointer to binary data buffer
198     writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
199 }
200 }

```

Для начала запустим программу так, чтобы фильтр возвращал EXCEPTION_EXECUTE_HANDLE. Это считается нормальной ситуацией, и на рисунке 13 видно как происходит передача управления.

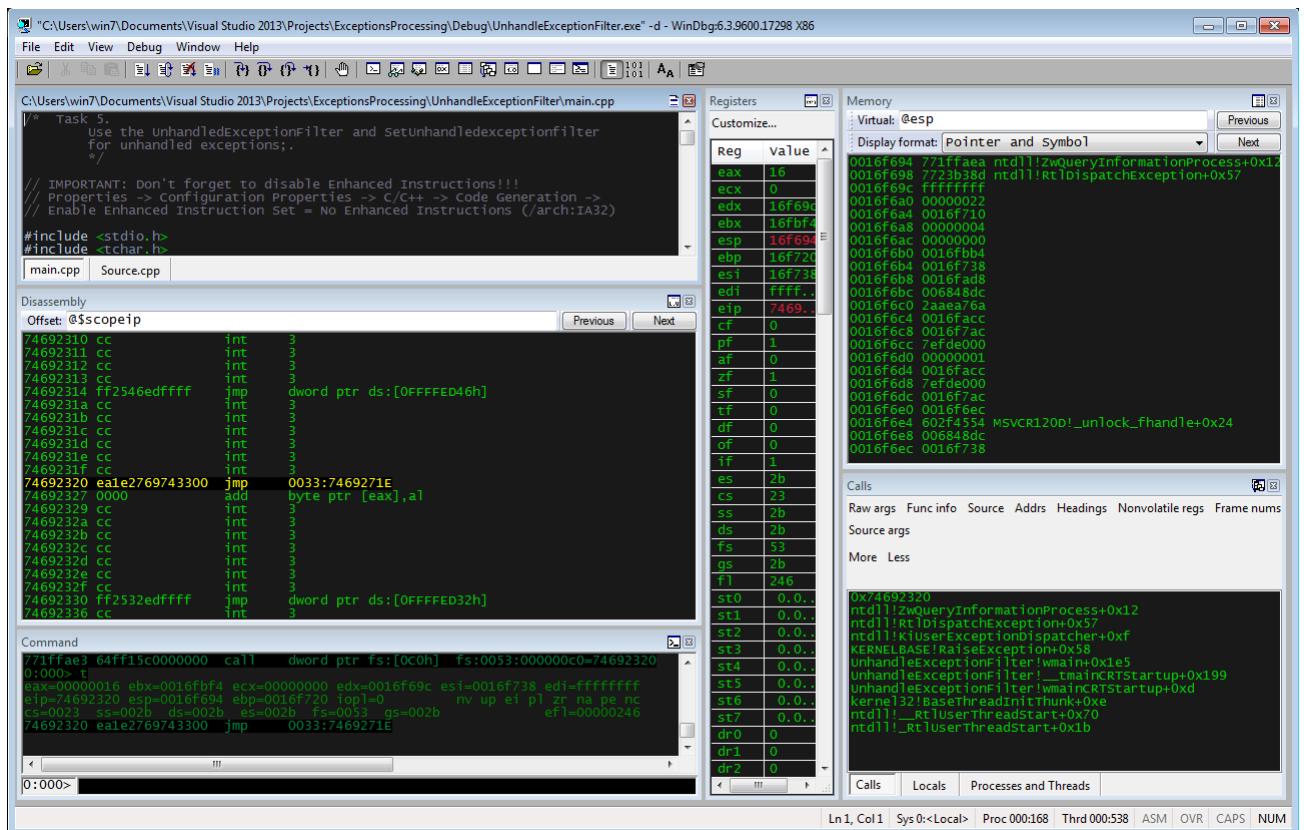


Рис. 13: Нормальная обработка исключения через фильтр

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.

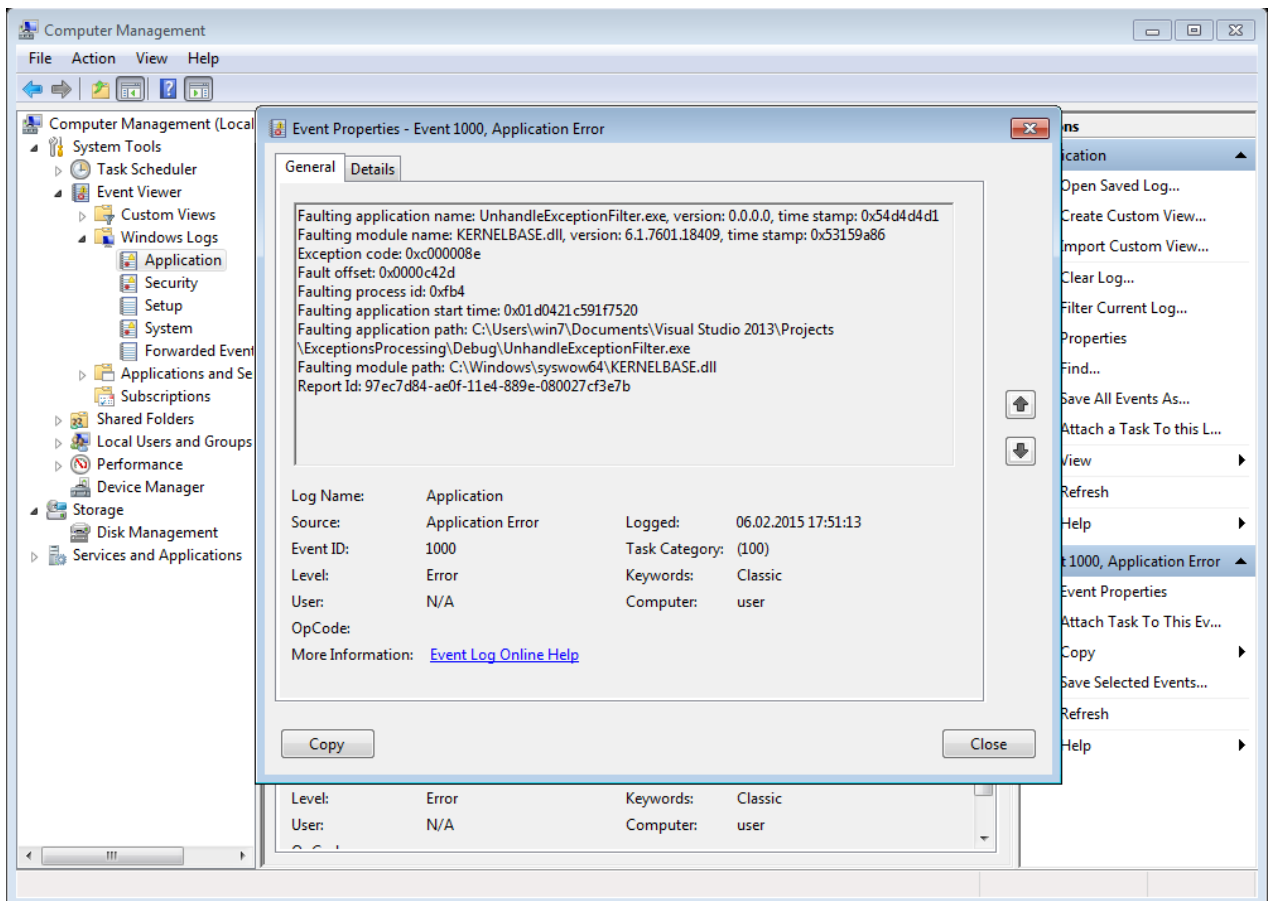


Рис. 14: Исключение зафиксировано в системном журнале

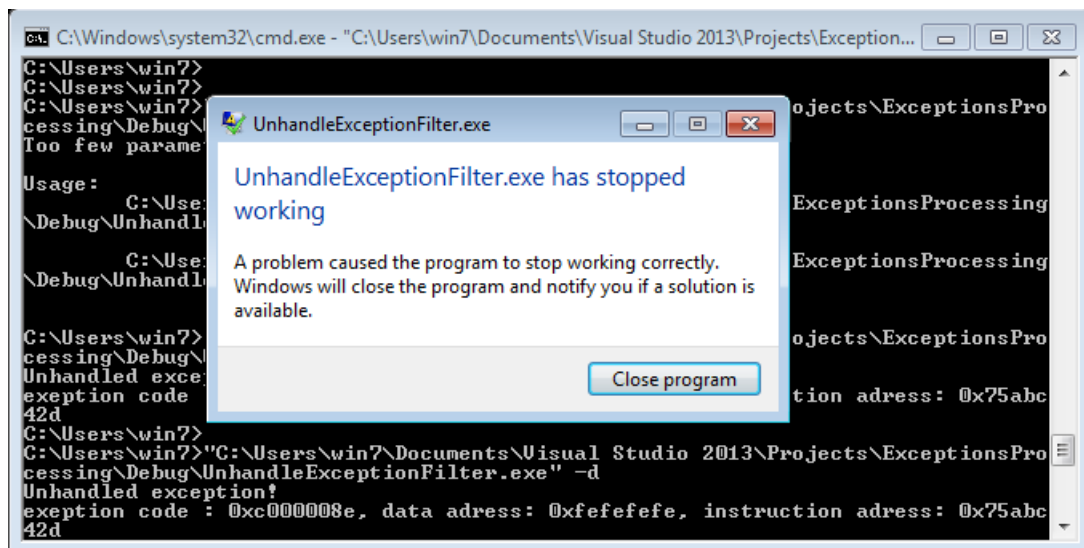


Рис. 15: Обработчик успел выполниться до системной ошибки

В логе программы можно прочитать информацию о произошедшей исключительной ситуации. 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\  
   ExceptionsProcessing\Debug\UnhandleExceptionFilter.exe is starting.  
2 [6/2/2015 18:9:43] Task: DIVIDE_BY_ZERO exception.  
3 [6/2/2015 18:9:43] Ready for generate DIVIDE_BY_ZERO exception.  
4 [6/2/2015 18:9:43] Unhandled exception!  
5 exeption code : 0xc000008e, data adress: 0xfefefefe, instruction adress: 0  
   x75abc42d
```

Из рассмотренного примера становится видно, что возврат `EXCEPTION_EXECUTE_HANDLER` является более предпочтительным 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  /* Task 6.
2    Nested exception process;
3    */
4
5  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!
6  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
7  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)
8
9  #include <stdio.h>
10 #include <tchar.h>
11 #include <cstring>
12 #include <cfloat>
13 #include <excpt.h>
14 #include <windows.h>
15 #include <time.h>
16
17 #include "messages.h"
18
19 // log
20 FILE* logfile;
21 HANDLE eventlog;
22
23 void initlog(const _TCHAR* prog);
24 void closelog();
25 void writelog(_TCHAR* format, ...);
26 void syslog(WORD category, WORD identifier, LPWSTR message);
27
```

```

28 // Defines the entry point for the console application.
29 int _tmain(int argc, _TCHAR* argv[]) {
30     //Init log
31     initlog(argv[0]);
32     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
33
34     // Floating point exceptions are masked by default.
35     _clearfp();
36     _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
37
38     __try {
39         __try {
40             writelog(_T("Ready for generate DIVIDE_BY_ZERO exception.));
41             syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
42                 _T("Ready for generate DIVIDE_BY_ZERO exception.));
43             RaiseException(EXCEPTION_FLT_DIVIDE_BY_ZERO,
44                 EXCEPTION_NONCONTINUABLE, 0, NULL);
45             writelog(_T("DIVIDE_BY_ZERO exception is generated.));
46         }
47         __except ((GetExceptionCode() == EXCEPTION_FLT_OVERFLOW) ?
48             EXCEPTION_EXECUTE_HANDLER :
49                 EXCEPTION_CONTINUE_SEARCH)
50         {
51             writelog(_T("Internal handler in action.));
52             _tprintf(_T("Internal handler in action.));
53             syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
54                 _T("Internal handler in action.));
55         }
56     }
57     __except ((GetExceptionCode() == EXCEPTION_FLT_DIVIDE_BY_ZERO) ?
58         EXCEPTION_EXECUTE_HANDLER :
59             EXCEPTION_CONTINUE_SEARCH)
60     {
61         writelog(_T("External handler in action.));
62         _tprintf(_T("External handler in action.));
63         syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
64             _T("Internal handler in action.));
65     }
66
67     closelog();
68     CloseHandle(eventlog);
69     exit(0);
70 }
71
72 void initlog(const _TCHAR* prog) {

```

```

73  _TCHAR logname[255];
74  wcsncpy_s(logname, prog);
75
76  // replace extension
77  _TCHAR* extension;
78  extension = wcsstr(logname, _T(".exe"));
79  wcsncpy_s(extension, 5, _T(".log"), 4);
80
81  // Try to open log file for append
82  if (_wfopen_s(&logfile, logname, _T("a+"))) {
83      _wprintf(_T("The following error occurred"));
84      _tprintf(_T("Can't open log file %s\n"), logname);
85      exit(1);
86  }
87
88  writelog(_T("%s is starting."), prog);
89 }
90
91 void closelog() {
92     writelog(_T("Shutting down.\n"));
93     fclose(logfile);
94 }
95
96 void writelog(_TCHAR* format, ...) {
97     _TCHAR buf[255];
98     va_list ap;
99
100    struct tm newtime;
101    __time64_t long_time;
102
103    // Get time as 64-bit integer.
104    _time64(&long_time);
105    // Convert to local time.
106    _localtime64_s(&newtime, &long_time);
107
108    // Convert to normal representation.
109    swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
110        newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
111        newtime.tm_min, newtime.tm_sec);
112
113    // Write date and time
114    fwprintf(logfile, _T("%s"), buf);
115    // Write all params
116    va_start(ap, format);
117    _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);

```

```

118     fprintf(logfile, _T("%s"), buf);
119     va_end(ap);
120     // New sting
121     fprintf(logfile, _T("\n"));
122 }
123
124 void syslog(WORD category, WORD identifier, LPWSTR message) {
125     LPWSTR pMessages[1] = { message };
126
127     if (!ReportEvent(
128         eventlog,           // event log handle
129         EVENTLOG_INFORMATION_TYPE, // event type
130         category,           // event category
131         identifier,         // event identifier
132         NULL,               // user security identifier
133         1,                  // number of substitution strings
134         0,                  // data size
135         (LPCWSTR*)pMessages, // pointer to strings
136         NULL)) {            // pointer to binary data buffer
137         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
138     }
139 }

```

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[1].

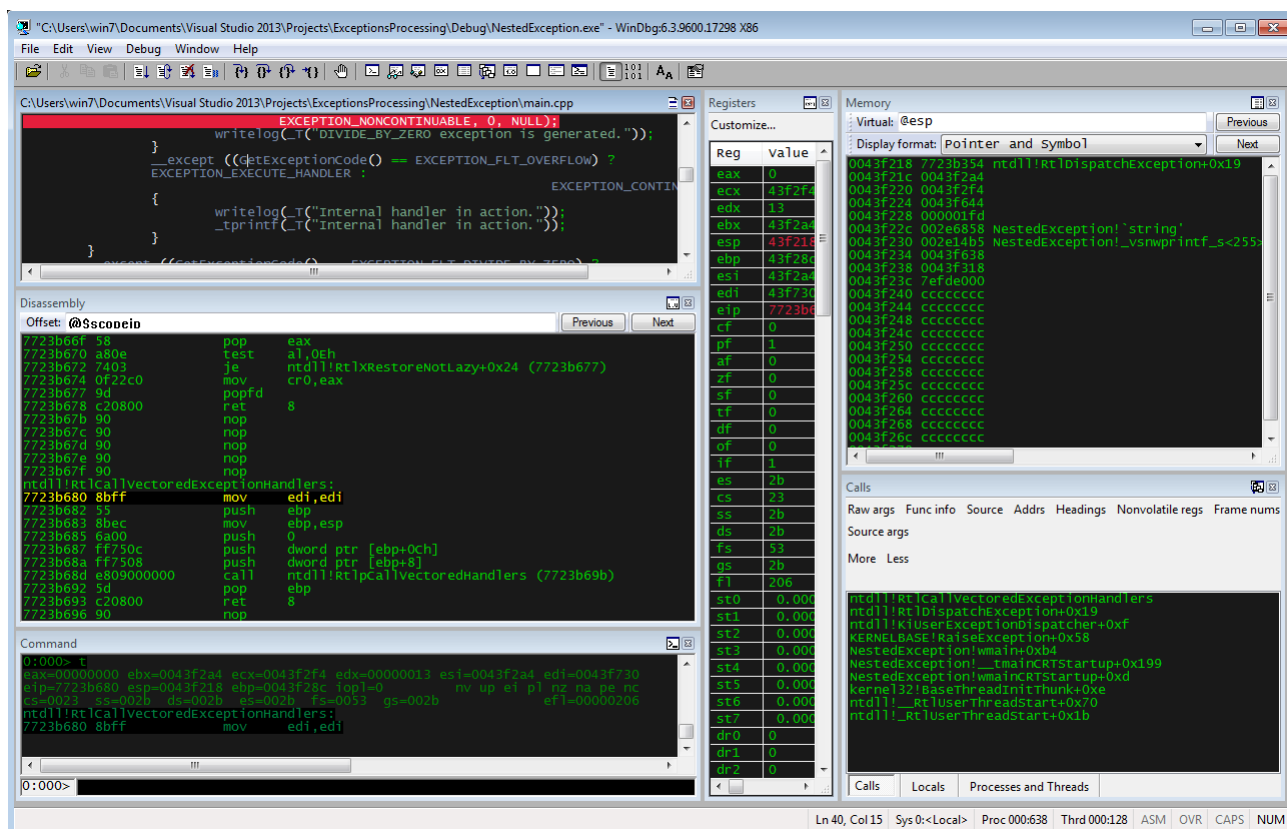


Рис. 16: Проверка условий обработчика исключения

При этом создаётся опасность утечки ресурсов, поэтому желательно обрабатывать исключительные 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\ExceptionsProcessing\Debug\NestedException.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[1].

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

```
1  /* Task 7.
2  Get out of the __try block by using the goto;
3  */
4
5  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!
6  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
7  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)
8
9  #include <stdio.h>
10 #include <tchar.h>
11 #include <cstring>
12 #include <cfloat>
13 #include <excpt.h>
14 #include <windows.h>
15 #include <time.h>
16
17 #include "messages.h"
18
19 // log
20 FILE* logfile;
21 HANDLE eventlog;
22
23 void initlog(const _TCHAR* prog);
24 void closelog();
25 void writelog(_TCHAR* format, ...);
26 void syslog(WORD category, WORD identifier, LPWSTR message);
27
28 // Defines the entry point for the console application.
```

```

29 int _tmain(int argc, _TCHAR* argv[]) {
30     //Init log
31     initlog(argv[0]);
32     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
33
34     // Floating point exceptions are masked by default.
35     _clearfp();
36     _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
37
38     __try {
39         writelog(_T("Call goto"));
40         goto OUT_POINT;
41         writelog(_T("Ready for generate DIVIDE_BY_ZERO exception.));
42         syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
43             _T("Ready for generate DIVIDE_BY_ZERO exception.));
44         RaiseException(EXCEPTION_FLT_DIVIDE_BY_ZERO,
45             EXCEPTION_NONCONTINUABLE, 0, NULL);
46         writelog(_T("DIVIDE_BY_ZERO exception is generated.));
47     }
48     __except (EXCEPTION_EXECUTE_HANDLER)
49     {
50         writelog(_T("Handler in action.));
51         _tprintf(_T("Handler in action.));
52         syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
53             _T("Handler in action.));
54     }
55 OUT_POINT:
56     writelog(_T("A point outside the __try block.));
57     _tprintf(_T("A point outside the __try block.));
58     syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
59         _T("A point outside the __try block.));
60
61     closelog();
62     CloseHandle(eventlog);
63     exit(0);
64 }
65
66 void initlog(const _TCHAR* prog) {
67     _TCHAR logname[255];
68     wcsncpy_s(logname, prog);
69
70     // replace extension
71     _TCHAR* extension;
72     extension = wcsstr(logname, _T(".exe"));
73     wcsncpy_s(extension, 5, _T(".log"), 4);

```

```

74
75 // Try to open log file for append
76 if (_wfopen_s(&logfile, logname, _T("a+"))) {
77     _wprintf(_T("The following error occurred"));
78     _tprintf(_T("Can't open log file %s\n"), logname);
79     exit(1);
80 }
81
82 writelog(_T("%s is starting."), prog);
83 }
84
85 void closelog() {
86     writelog(_T("Shutting down.\n"));
87     fclose(logfile);
88 }
89
90 void writelog(_TCHAR* format, ...) {
91     _TCHAR buf[255];
92     va_list ap;
93
94     struct tm newtime;
95     __time64_t long_time;
96
97     // Get time as 64-bit integer.
98     _time64(&long_time);
99     // Convert to local time.
100     _localtime64_s(&newtime, &long_time);
101
102     // Convert to normal representation.
103     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
104         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
105         newtime.tm_min, newtime.tm_sec);
106
107     // Write date and time
108     fwprintf(logfile, _T("%s"), buf);
109     // Write all params
110     va_start(ap, format);
111     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
112     fwprintf(logfile, _T("%s"), buf);
113     va_end(ap);
114     // New sting
115     fwprintf(logfile, _T("\n"));
116 }
117
118 void syslog(WORD category, WORD identifier, LPWSTR message) {

```

```

119 LPWSTR pMessages[1] = { message };
120
121 if (!ReportEvent(
122     eventlog,          // event log handle
123     EVENTLOG_INFORMATION_TYPE, // event type
124     category,          // event category
125     identifier,        // event identifier
126     NULL,              // user security identifier
127     1,                 // number of substitution strings
128     0,                 // data size
129     (LPCWSTR*)pMessages, // pointer to strings
130     NULL)) {           // pointer to binary data buffer
131     writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
132 }
133 }

```

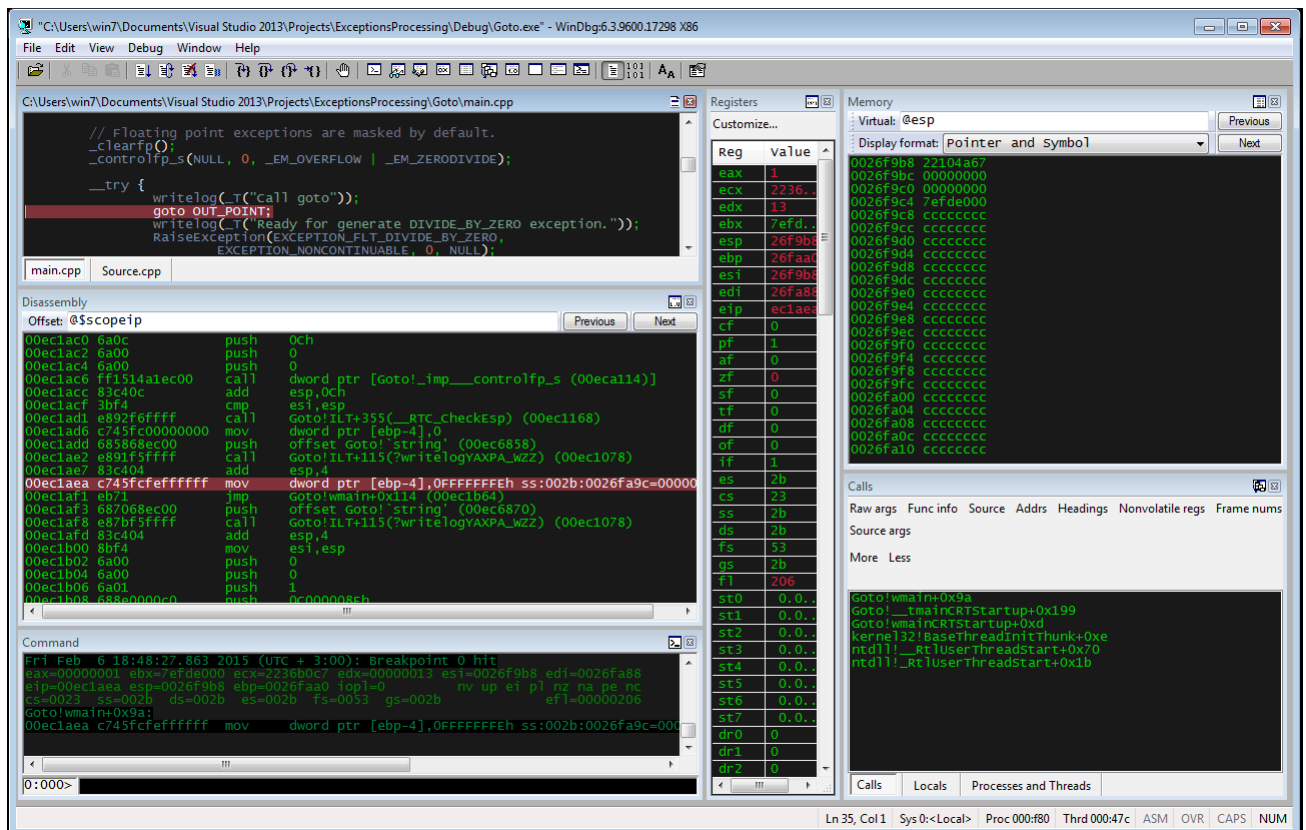


Рис. 17: Переход по goto

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 логге 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\  
    ExceptionsProcessing\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[1].

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

```
1  /* Task 8.
2  Get out of the __try block by using the leave;
3  */
4
5  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!
6  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
7  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)
8
9  #include <stdio.h>
10 #include <tchar.h>
11 #include <cstring>
12 #include <cfloat>
13 #include <excpt.h>
14 #include <windows.h>
15 #include <time.h>
16
17 #include "messages.h"
18
19 // log
20 FILE* logfile;
21 HANDLE eventlog;
22
23 void initlog(const _TCHAR* prog);
24 void closelog();
25 void writelog(_TCHAR* format, ...);
26 void syslog(WORD category, WORD identifier, LPWSTR message);
27
28 // Defines the entry point for the console application.
```

```

29 int _tmain(int argc, _TCHAR* argv[]) {
30     //Init log
31     initlog(argv[0]);
32     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
33
34     // Floating point exceptions are masked by default.
35     _clearfp();
36     _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
37
38     __try {
39         writelog(_T("Call __leave"));
40         __leave;
41         writelog(_T("Ready for generate DIVIDE_BY_ZERO exception.));
42         syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
43             _T("Ready for generate DIVIDE_BY_ZERO exception.));
44         RaiseException(EXCEPTION_FLT_DIVIDE_BY_ZERO,
45             EXCEPTION_NONCONTINUABLE, 0, NULL);
46         writelog(_T("DIVIDE_BY_ZERO exception is generated.));
47     }
48     __except (EXCEPTION_EXECUTE_HANDLER)
49     {
50         writelog(_T("Handler in action.));
51         _tprintf(_T("Handler in action.));
52         syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
53             _T("Handler in action.));
54     }
55     writelog(_T("A point outside the __try block.));
56     _tprintf(_T("A point outside the __try block.));
57     syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
58         _T("A point outside the __try block.));
59
60     closelog();
61     CloseHandle(eventlog);
62     exit(0);
63 }
64
65 void initlog(const _TCHAR* prog) {
66     _TCHAR logname[255];
67     wcscpy_s(logname, prog);
68
69     // replace extension
70     _TCHAR* extension;
71     extension = wcsstr(logname, _T(".exe"));
72     wcsncpy_s(extension, 5, _T(".log"), 4);
73

```

```

74 // Try to open log file for append
75 if (_wfopen_s(&logfile, logname, _T("a+"))) {
76     _wpperror(_T("The following error occurred"));
77     _tprintf(_T("Can't open log file %s\n"), logname);
78     exit(1);
79 }
80
81 writelog(_T("%s is starting."), prog);
82 }
83
84 void closelog() {
85     writelog(_T("Shutting down.\n"));
86     fclose(logfile);
87 }
88
89 void writelog(_TCHAR* format, ...) {
90     _TCHAR buf[255];
91     va_list ap;
92
93     struct tm newtime;
94     __time64_t long_time;
95
96     // Get time as 64-bit integer.
97     _time64(&long_time);
98     // Convert to local time.
99     _localtime64_s(&newtime, &long_time);
100
101     // Convert to normal representation.
102     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
103         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
104         newtime.tm_min, newtime.tm_sec);
105
106     // Write date and time
107     fwprintf(logfile, _T("%s"), buf);
108     // Write all params
109     va_start(ap, format);
110     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
111     fwprintf(logfile, _T("%s"), buf);
112     va_end(ap);
113     // New sting
114     fwprintf(logfile, _T("\n"));
115 }
116
117 void syslog(WORD category, WORD identifier, LPWSTR message) {
118     LPWSTR pMessages[1] = { message };

```



```

119
120 if (!ReportEvent(
121     eventlog,           // event log handle
122     EVENTLOG_INFORMATION_TYPE, // event type
123     category,          // event category
124     identifier,         // event identifier
125     NULL,               // user security identifier
126     1,                  // number of substitution strings
127     0,                  // data size
128     (LPCWSTR*)pMessages, // pointer to strings
129     NULL)) {            // pointer to binary data buffer
130     writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
131 }
132 }

```

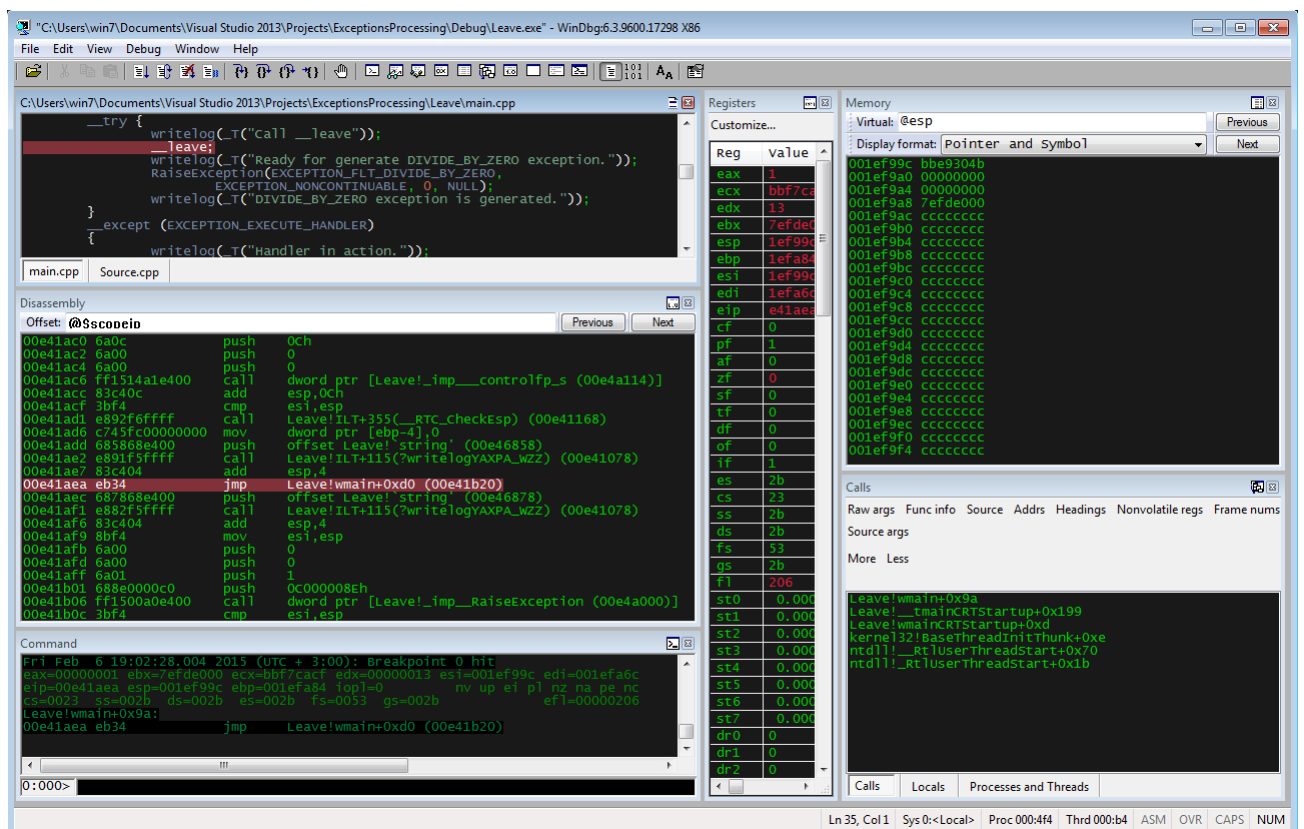


Рис. 18: Переход по __leave

Листинг 18: Переход по оператору __leave

```

1 [6/2/2015 19:8:37] C:\Users\win7\Documents\Visual Studio 2013\Projects\
  ExceptionsProcessing\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 результата 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/C++. Для этого необходимо включить соответствующие опции в компиляторе (/EHa)[1, 4].

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

```
1  /* Task 9.
2     Convert structural exceptions to the C language exceptions,
3     using the translator;
4     */
5
6  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!
7  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
8  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)
9
10 // IMPORTANT: Don't forget to enable SEH!!!
11 // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
12 // Enable C++ Exceptions = Yes with SEH Exceptions (/EHa)
13
14 #include <stdio.h>
15 #include <tchar.h>
16 #include <cstring>
17 #include <cfloat>
18 #include <stdexcept>
19 #include <excpt.h>
20 #include <windows.h>
21 #include <time.h>
22
23 #include "messages.h"
24
25 // log
26 FILE* logfile;
27 HANDLE eventlog;
28
```

```

29 void initlog(const _TCHAR* prog);
30 void closelog();
31 void writelog(_TCHAR* format, ...);
32 void syslog(WORD category, WORD identifier, LPWSTR message);
33
34 void translator(unsigned int u, EXCEPTION_POINTERS* pExp);
35
36 // My exception
37 class translator_exception {
38 public:
39     translator_exception(const wchar_t* str) {
40         wcsncpy_s(buf, sizeof(buf), str, sizeof(buf));
41     }
42     const wchar_t* what() { return buf; }
43 private:
44     wchar_t buf[255];
45 };
46
47 // Defines the entry point for the console application.
48 int _tmain(int argc, _TCHAR* argv[]) {
49     //Init log
50     initlog(argv[0]);
51     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
52
53     // Floating point exceptions are masked by default.
54     _clearfp();
55     _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
56
57     try {
58         writelog(_T("Ready for translator ativation."));
59         _set_se_translator(translator);
60         writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
61         syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
62             _T("Ready for generate DIVIDE_BY_ZERO exception.));
63         RaiseException(EXCEPTION_FLT_DIVIDE_BY_ZERO,
64             EXCEPTION_NONCONTINUABLE, 0, NULL);
65         writelog(_T("DIVIDE_BY_ZERO exception is generated.));
66     }
67     catch (translator_exception &e) {
68         _tprintf(_T("CPP exception: %s"), e.what());
69         writelog(_T("CPP exception: %s"), e.what());
70         syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION, _T("CPP exception"));
71     }
72
73     closelog();

```

```

74 CloseHandle(eventlog);
75 exit(0);
76 }
77
78 void translator(unsigned int u, EXCEPTION_POINTERS* pExp) {
79     writelog(_T("Translator in action."));
80     if (u == EXCEPTION_FLT_DIVIDE_BY_ZERO)
81         throw translator_exception(_T("EXCEPTION_FLT_DIVIDE_BY_ZERO"));
82 }
83
84 void initlog(const _TCHAR* prog) {
85     _TCHAR logname[255];
86     wcsncpy_s(logname, prog);
87
88     // replace extension
89     _TCHAR* extension;
90     extension = wcsstr(logname, _T(".exe"));
91     wcsncpy_s(extension, 5, _T(".log"), 4);
92
93     // Try to open log file for append
94     if (_wfopen_s(&logfile, logname, _T("a+"))) {
95         _wpperror(_T("The following error occurred"));
96         _tprintf(_T("Can't open log file %s\n"), logname);
97         exit(1);
98     }
99
100     writelog(_T("%s is starting."), prog);
101 }
102
103 void closelog() {
104     writelog(_T("Shutting down.\n"));
105     fclose(logfile);
106 }
107
108 void writelog(_TCHAR* format, ...) {
109     _TCHAR buf[255];
110     va_list ap;
111
112     struct tm newtime;
113     __time64_t long_time;
114
115     // Get time as 64-bit integer.
116     _time64(&long_time);
117     // Convert to local time.
118     _localtime64_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     newtime.tm_min, newtime.tm_sec);
124
125 // Write date and time
126 fwprintf(logfile, _T("%s"), buf);
127 // Write all params
128 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 void syslog(WORD category, WORD identifier, LPWSTR message) {
137     LPWSTR pMessages[1] = { message };
138
139     if (!ReportEvent(
140         eventlog,           // event log handle
141         EVENTLOG_INFORMATION_TYPE, // event type
142         category,          // event category
143         identifier,        // event identifier
144         NULL,              // user security identifier
145         1,                 // number of substitution strings
146         0,                 // data size
147         (LPCWSTR*)pMessages, // pointer to strings
148         NULL)) {           // pointer to binary data buffer
149         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
150     }
151 }

```

На рисунке 19 видна передача управления от генерации исключения в ядре к созданию пользовательского исключения. Листинг 20 показывает, какие участки кода были задействованы и в каком порядке.

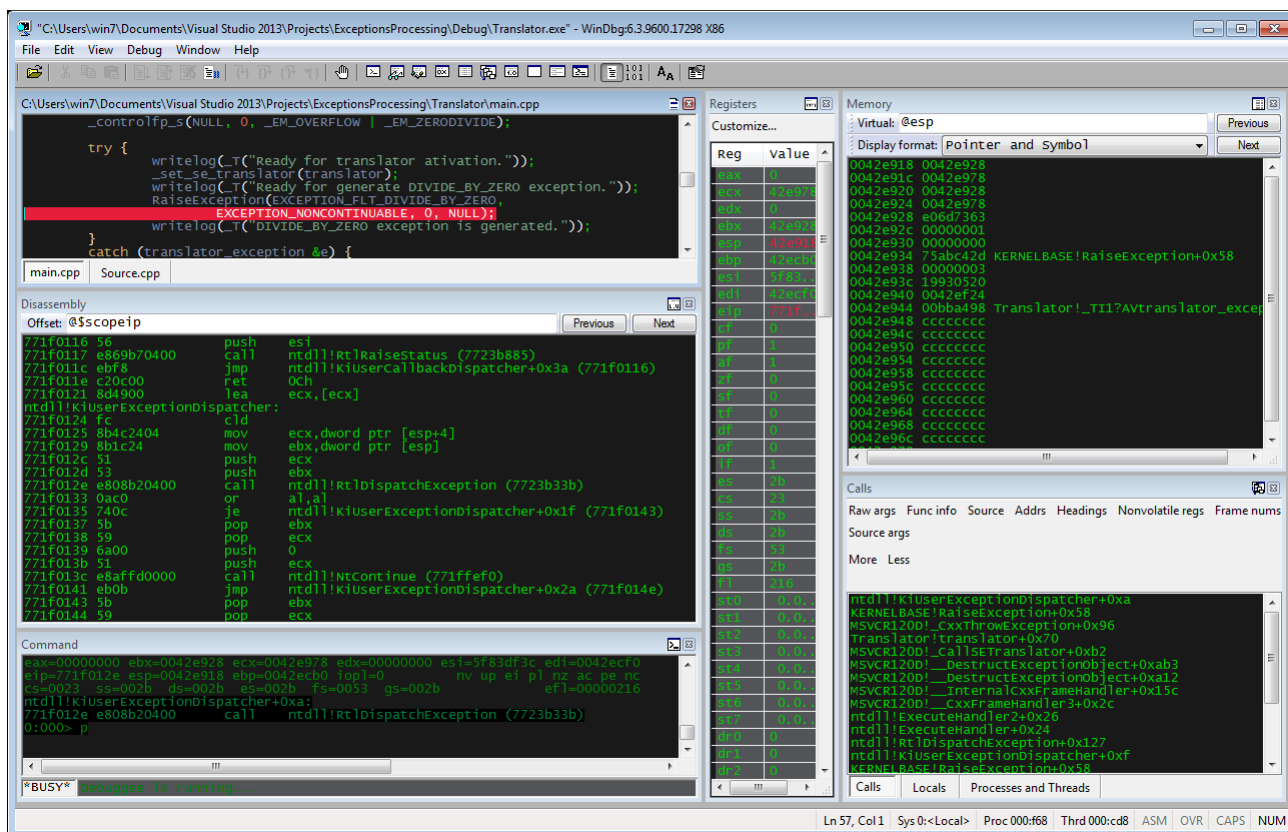


Рис. 19: Передача управления коду создания пользовательского исключения

Листинг 20: Результат работы Translator.exe

```

1 [6/2/2015 19:33:15] C:\Users\win7\Documents\Visual Studio 2013\Projects\
  ExceptionsProcessing\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 C++-исключение (в данном случае используется собственный класс исключения, определённый 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[1].

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

```
1  /* Task 10.
2     Use the final handler finally;
3     */
4
5  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!
6  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->
7  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)
8
9  #include <stdio.h>
10 #include <tchar.h>
11 #include <cfloat>
12 #include <except.h>
13 #include <time.h>
14 #include <windows.h>
15
16 #include "messages.h"
17
18 // log
19 FILE* logfile;
20 HANDLE eventlog;
21
22 void initlog(const _TCHAR* prog);
23 void closelog();
24 void writelog(_TCHAR* format, ...);
25 void syslog(WORD category, WORD identifier, LPWSTR message);
26
27 // Defines the entry point for the console application.
28 int _tmain(int argc, _TCHAR* argv[]) {
29     //Init log
30     initlog(argv[0]);
```



```

31 eventlog = RegisterEventSource(NULL, L"MyEventProvider");
32
33 // Floating point exceptions are masked by default.
34 _clearfp();
35 _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
36
37 __try {
38     // No exception
39     writelog(_T("Try block.));
40 }
41 __finally
42 {
43     writelog(_T("There is no exception, but the handler is called.));
44     _tprintf(_T("There is no exception, but the handler is called.));
45     syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
46         _T("There is no exception, but the handler is called.));
47 }
48
49 closelog();
50 CloseHandle(eventlog);
51 exit(0);
52 }
53
54 void initlog(const _TCHAR* prog) {
55     _TCHAR logname[255];
56     wcscpy_s(logname, prog);
57
58     // replace extension
59     _TCHAR* extension;
60     extension = wcsstr(logname, _T(".exe"));
61     wcsncpy_s(extension, 5, _T(".log"), 4);
62
63     // Try to open log file for append
64     if (_wfopen_s(&logfile, logname, _T("a+"))) {
65         _wprintf(_T("The following error occurred"));
66         _tprintf(_T("Can't open log file %s\n"), logname);
67         exit(1);
68     }
69
70     writelog(_T("%s is starting."), prog);
71 }
72
73 void closelog() {
74     writelog(_T("Shutting down.\n"));
75     fclose(logfile);

```

```

76 }
77
78 void writelog(_TCHAR* format, ...) {
79     _TCHAR buf[255];
80     va_list ap;
81
82     struct tm newtime;
83     __time64_t long_time;
84
85     // Get time as 64-bit integer.
86     _time64(&long_time);
87     // Convert to local time.
88     _localtime64_s(&newtime, &long_time);
89
90     // Convert to normal representation.
91     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
92         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
93         newtime.tm_min, newtime.tm_sec);
94
95     // Write date and time
96     fwprintf(logfile, _T("%s"), buf);
97     // Write all params
98     va_start(ap, format);
99     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
100    fwprintf(logfile, _T("%s"), buf);
101    va_end(ap);
102    // New sting
103    fwprintf(logfile, _T("\n"));
104 }
105
106 void syslog(WORD category, WORD identifier, LPWSTR message) {
107     LPWSTR pMessages[1] = { message };
108
109     if (!ReportEvent(
110         eventlog,          // event log handle
111         EVENTLOG_INFORMATION_TYPE, // event type
112         category,          // event category
113         identifier,        // event identifier
114         NULL,              // user security identifier
115         1,                 // number of substitution strings
116         0,                 // data size
117         (LPCWSTR*)pMessages, // pointer to strings
118         NULL)) {           // pointer to binary data buffer
119         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
120     }

```

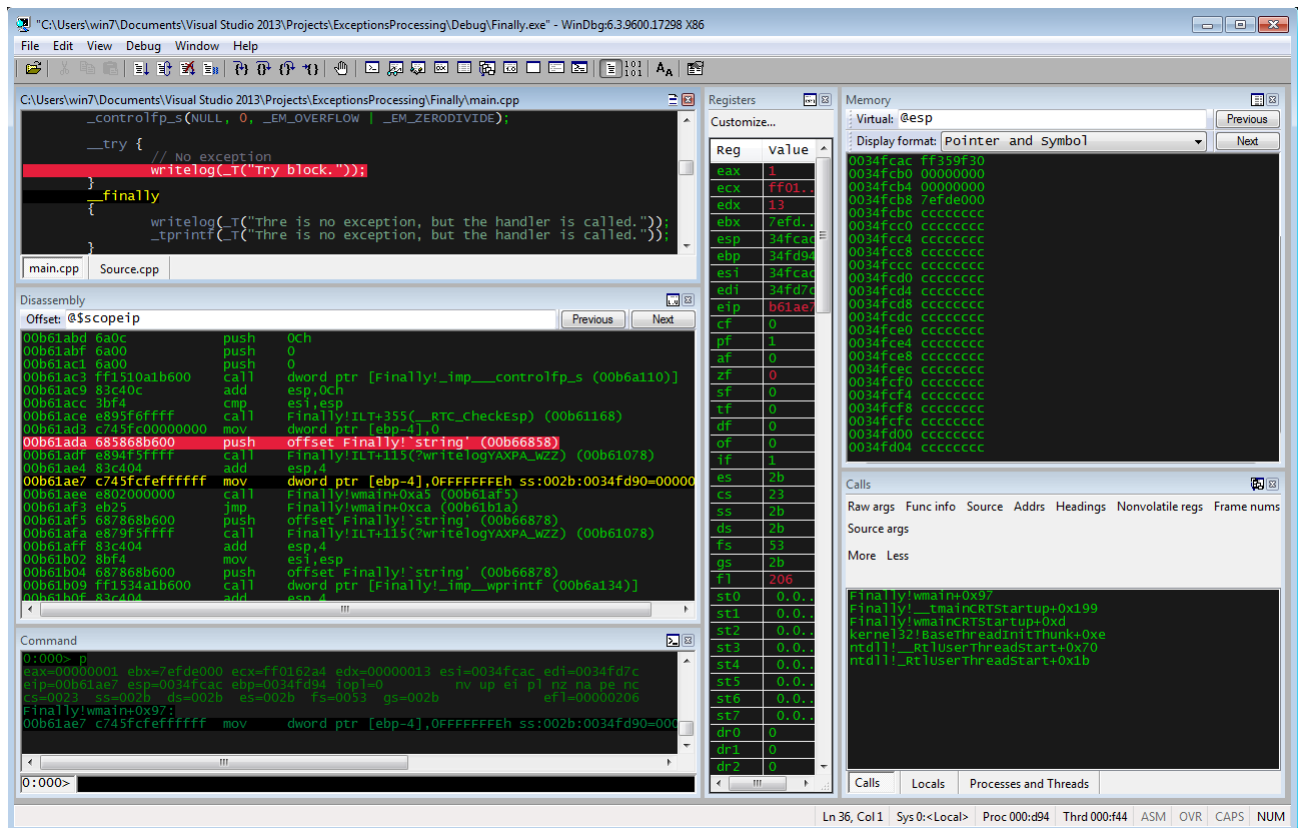


Рис. 20: Переход в блок finally

Листинг 22: Результат работы Finally.exe

- 1 [6/2/2015 19:42:27] C:\Users\win7\Documents\Visual Studio 2013\Projects\ExceptionsProcessing\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[1].

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

```
1  /* Task 11.  
2  Check the correctness of the exit from the __try block using  
3  the AbnormalTermination function in the final handler finally.  
4  */  
5  
6  // IMPORTANT: Don't forget to disable Enhanced Instructions!!!  
7  // Properties -> Configuration Properties -> C/C++ -> Code Generation ->  
8  // Enable Enhanced Instruction Set = No Enhanced Instructions (/arch:IA32)  
9  
10 #include <stdio.h>  
11 #include <tchar.h>  
12 #include <cstring>  
13 #include <cfloat>  
14 #include <except.h>  
15 #include <windows.h>  
16 #include <time.h>  
17  
18 #include "messages.h"  
19  
20 // log  
21 FILE* logfile;  
22 HANDLE eventlog;  
23  
24 void initlog(const _TCHAR* prog);
```

```

25 void closelog();
26 void writelog(_TCHAR* format, ...);
27 void syslog(WORD category, WORD identifier, LPWSTR message);
28
29 // Defines the entry point for the console application.
30 int _tmain(int argc, _TCHAR* argv[]) {
31     //Init log
32     initlog(argv[0]);
33     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
34
35     // Floating point exceptions are masked by default.
36     _clearfp();
37     _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
38
39     __try {
40         writelog(_T("Call goto"));
41         goto OUT_POINT;
42         writelog(_T("Ready for generate DIVIDE_BY_ZERO exception.));
43         syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
44             _T("Ready for generate DIVIDE_BY_ZERO exception.));
45         RaiseException(EXCEPTION_FLT_DIVIDE_BY_ZERO,
46             EXCEPTION_NONCONTINUABLE, 0, NULL);
47         writelog(_T("DIVIDE_BY_ZERO exception is generated.));
48     }
49     __finally
50     {
51         if (AbnormalTermination()) {
52             writelog(_T("%s"), _T("Abnormal termination in goto case.));
53             _tprintf(_T("%s"), _T("Abnormal termination in goto case.\n"));
54             syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
55                 _T("Abnormal termination in goto case.));
56         }
57         else {
58             writelog(_T("%s"), _T("Normal termination in goto case.));
59             _tprintf(_T("%s"), _T("Normal termination in goto case.\n"));
60             syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
61                 _T("Normal termination in goto case.));
62         }
63     }
64 OUT_POINT:
65     writelog(_T("A point outside the first __try block.));
66
67     __try {
68         writelog(_T("Call __leave"));
69         __leave;

```

```

70     writelog(_T("Ready for generate EXCEPTION_FLT_DIVIDE_BY_ZERO exception."
71         ));
72     syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
73         _T("Ready for generate FLT_OVERFLOW exception."));
74     RaiseException(EXCEPTION_FLT_DIVIDE_BY_ZERO,
75         EXCEPTION_NONCONTINUABLE, 0, NULL);
76     writelog(_T("EXCEPTION_FLT_DIVIDE_BY_ZERO exception is generated."));
77 }
78 __finally
79 {
80     if (AbnormalTermination()) {
81         writelog(_T("%s"), _T("Abnormal termination in __leave case."));
82         _tprintf(_T("%s"), _T("Abnormal termination in __leave case.\n"));
83         syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
84             _T("Abnormal termination in __leave case."));
85     }
86     else {
87         writelog(_T("%s"), _T("Normal termination in __leave case."));
88         _tprintf(_T("%s"), _T("Normal termination in __leave case.\n"));
89         syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
90             _T("Normal termination in __leave case."));
91     }
92     writelog(_T("A point outside the second __try block."));
93
94     closelog();
95     CloseHandle(eventlog);
96     exit(0);
97 }
98
99 void initlog(const _TCHAR* prog) {
100     _TCHAR logname[255];
101     wcsncpy_s(logname, prog);
102
103     // replace extension
104     _TCHAR* extension;
105     extension = wcsstr(logname, _T(".exe"));
106     wcsncpy_s(extension, 5, _T(".log"), 4);
107
108     // Try to open log file for append
109     if (_wfopen_s(&logfile, logname, _T("a+"))) {
110         _wprintf(_T("The following error occurred"));
111         _tprintf(_T("Can't open log file %s\n"), logname);
112         exit(1);
113     }

```

```

114
115     writelog(_T("%s is starting."), prog);
116 }
117
118 void closelog() {
119     writelog(_T("Shutting down.\n"));
120     fclose(logfile);
121 }
122
123 void writelog(_TCHAR* format, ...) {
124     _TCHAR buf[255];
125     va_list ap;
126
127     struct tm newtime;
128     __time64_t long_time;
129
130     // Get time as 64-bit integer.
131     _time64(&long_time);
132     // Convert to local time.
133     _localtime64_s(&newtime, &long_time);
134
135     // Convert to normal representation.
136     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
137         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
138         newtime.tm_min, newtime.tm_sec);
139
140     // Write date and time
141     fwprintf(logfile, _T("%s"), buf);
142     // Write all params
143     va_start(ap, format);
144     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
145     fwprintf(logfile, _T("%s"), buf);
146     va_end(ap);
147     // New sting
148     fwprintf(logfile, _T("\n"));
149 }
150
151 void syslog(WORD category, WORD identifier, LPWSTR message) {
152     LPWSTR pMessages[1] = { message };
153
154     if (!ReportEvent(
155         eventlog,           // event log handle
156         EVENTLOG_INFORMATION_TYPE, // event type
157         category,           // event category
158         identifier,         // event identifier

```

```

159     NULL,           // user security identifier
160     1,              // number of substitution strings
161     0,              // data size
162     (LPCWSTR*)pMessages, // pointer to strings
163     NULL)) {        // pointer to binary data buffer
164     writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
165 }
166 }

```

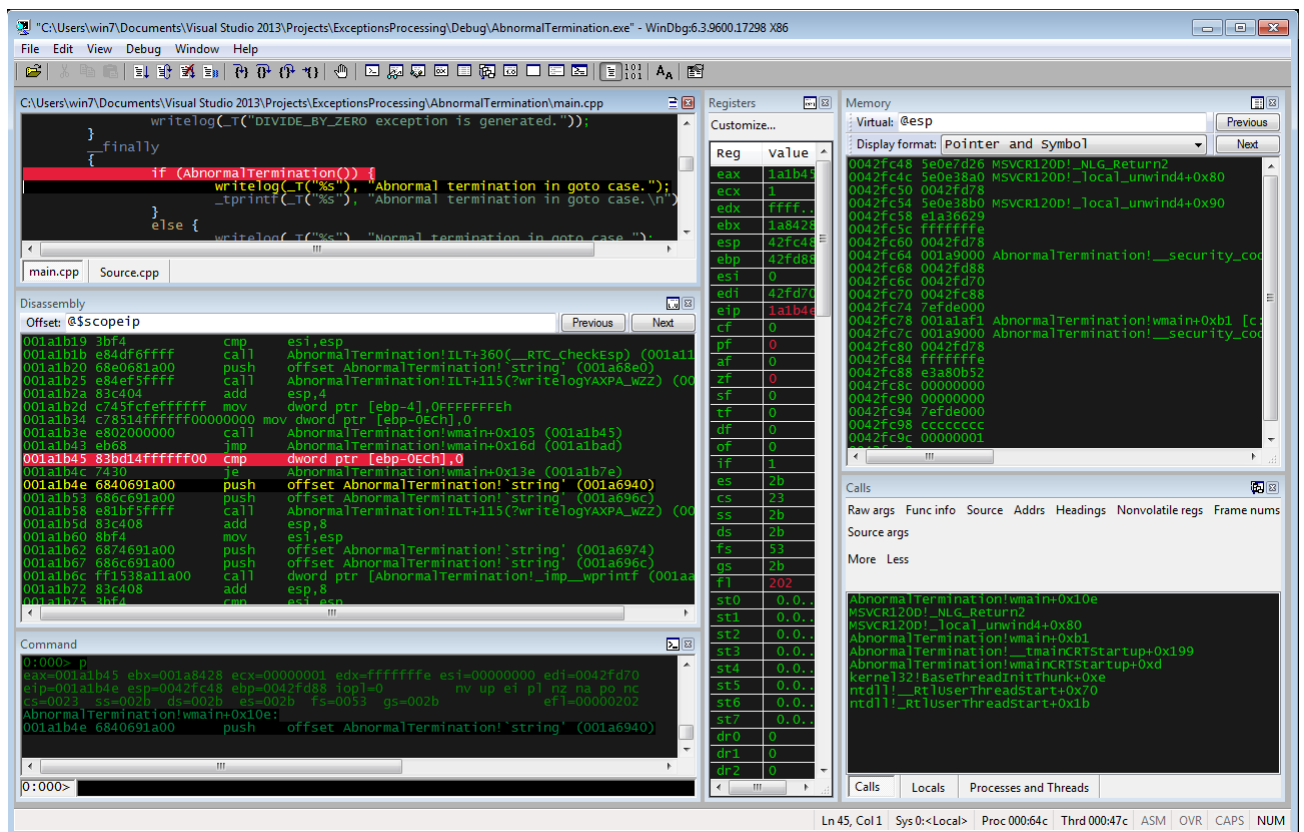


Рис. 21: Выход из защищаемого блока по goto

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.

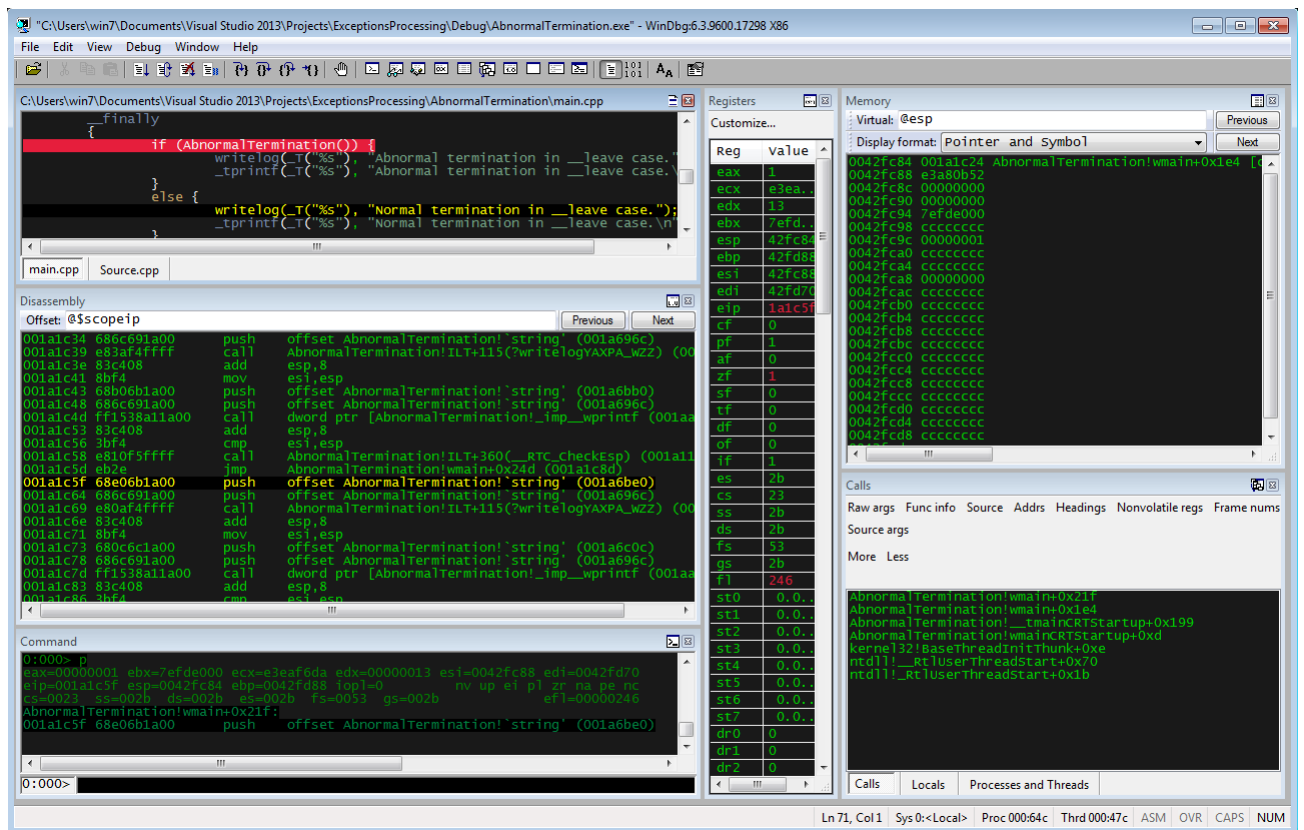


Рис. 22: Выход из защищаемого блока по `__leave`

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\
  ExceptionsProcessing\Debug\AbnormalTermination.exe is starting.
2 [6/2/2015 19:58:19] Call goto
3 [6/2/2015 19:58:19] Abnormal termination in goto case.
4 [6/2/2015 19:58:19] A point outside the first __try block.
5 [6/2/2015 19:58:19] Call __leave
6 [6/2/2015 19:58:19] Normal termination in __leave case.
7 [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 C++ используются ключевые слова `catch` и `throw`, а сам механизм исключений реализован с использованием SEH. Тем не менее, обработка исключений в C++ и SEH — это разные вещи. Их совместное применение требует внимательного обращения, 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 C++ и ограничиться применением в них только обработчиков исключений C++.

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

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 уровне языка C++ это разные инструменты, требующие разного подхода.

Литература

1. Конспект лекций Душутиной Е.В. по пред. "Системное программное обеспечение"
2. MSDN: Какие сведения содержатся в журналах событий? (Просмотр событий) – <http://windows.microsoft.com/ru-ru/windows/what-information-event-logs-event-viewer>
3. MSDN: ReportEvent function – <https://msdn.microsoft.com/aa363679.aspx>
4. MSDN: Structured Exception Handling – <https://msdn.microsoft.com/ms680657.aspx>