Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий

**Кафедра компьютерных систем и программных технологий**

**Отчёт по лабораторной работе №1 часть 1**
**Курс:** Системное программирование
**Тема:** Анализ обработки исключений в Windows

Выполнил студент группы 13541/3 _____ Д.В. Круминьш
(подпись)

Преподаватель _____ Е.В. Душутина
(подпись)

Санкт-Петербург
2017 г.

# Содержание

# Постановка задачи

1. Сгенерировать и обработать исключения с помощью функций WinAPI;
2. Получить код исключения с помощью функции GetExceptionCode:
   (a) Использовать эту функцию в выражении фильтра;
   (b) 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 UnhandleExceptionFilter и SetUnhadleExceptionFilter 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).

# Подготовка к выполнению работы

## 2.1 Введение

Исключение – это аномальное поведение во время выполнения, в случае отсутствия обработки исключений их возникновение приведет к немедленному прекращению выполнения программы. В операционной системе Microsoft Windows, механизмом обработки программных и аппаратных исключений является **SEH (Structured Exception Handling)**.

SEH предоставляет возможность определить блок программного кода, или обработчик исключений (exception handler).

Этот механизм обработки исключений отличается от обработки исключений в C++. Он разработан специально для Windows и реализован в Visual C++. Общая идея обработки похожа. Код заключается в блок обработки _try, но в отличии от C++ обработки дальше может следовать один из двух блоков обработки, это либо _finaly либо _except блок.

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 логирования отклика системы будет использоваться:

1. вывод результатов работы в консоли;
2. запись протокола работы в отдельный лог файл;
3. фиксирование события в системном журнале событий Windows.

## 2.2 Журнал событий

Журнал событий Windows - это средство, позволяющее программам и самой системе Windows регистрировать и хранить уведомления в одном месте. В журнале регистрируются все ошибки, информационные сообщения и предупреждения программ.

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.

Для просмотра журнала событий необходимо:

1. нажать кнопку **Пуск**;
2. выбрать **Панель управления**;
3. выбрать **Администрирование**;
4. выбрать **Просмотр событий**;
5. выбрать интересующий журнал, в данной работе будет использоваться журнал **Приложение**, категории **Журналы Windows**;
6. для просмотра события, кликнуть по нему.

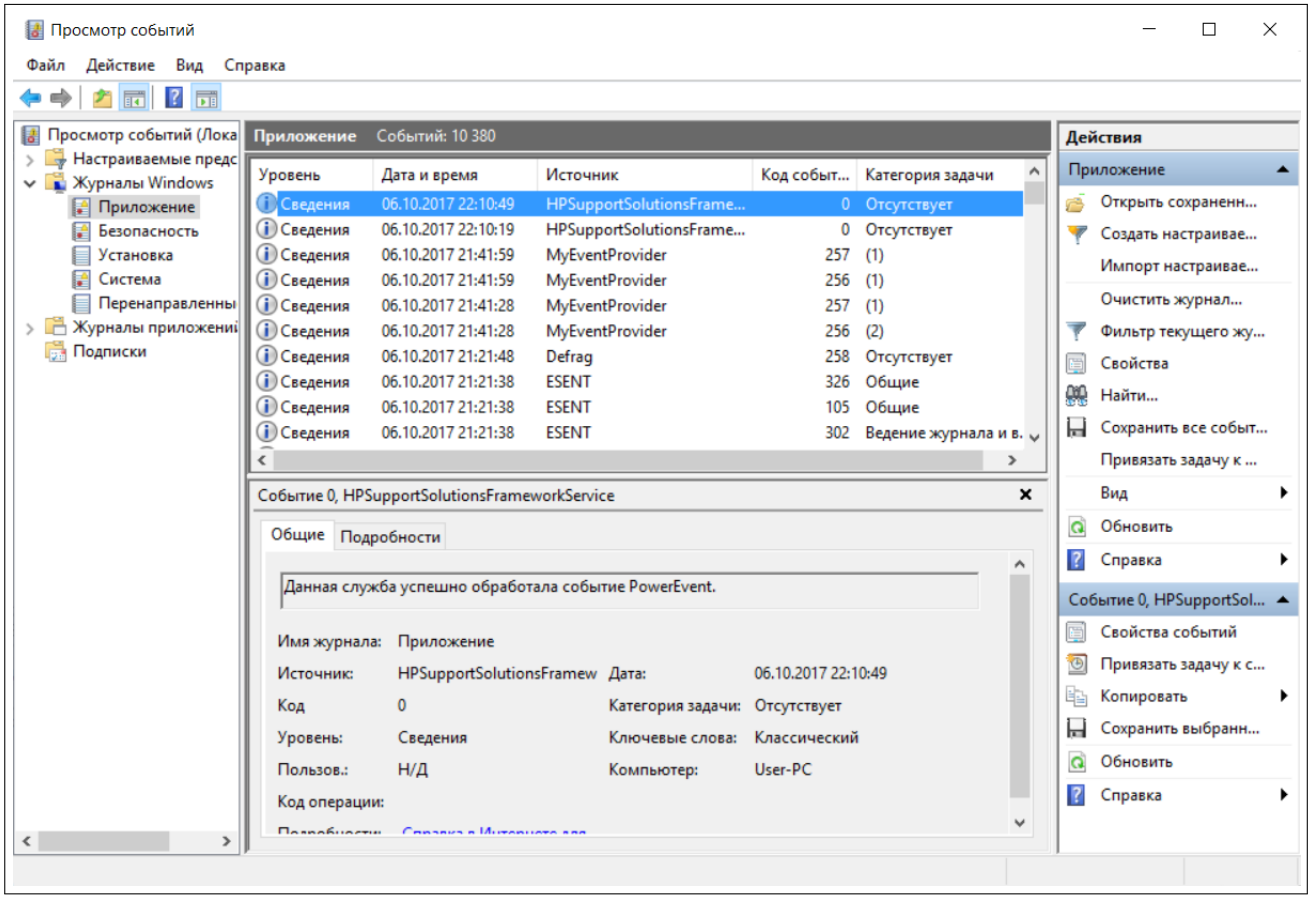



Рис. 2.1: Окно журнала событий

## 2.3 Настройка логирования

Для создания ресурс-файла нужно в меню проекта(вкладки **файлы ресурсов**) вызвать добавление нового файла, указать его тип (текстовый файл) и имя (в данном случае это messages.mc).

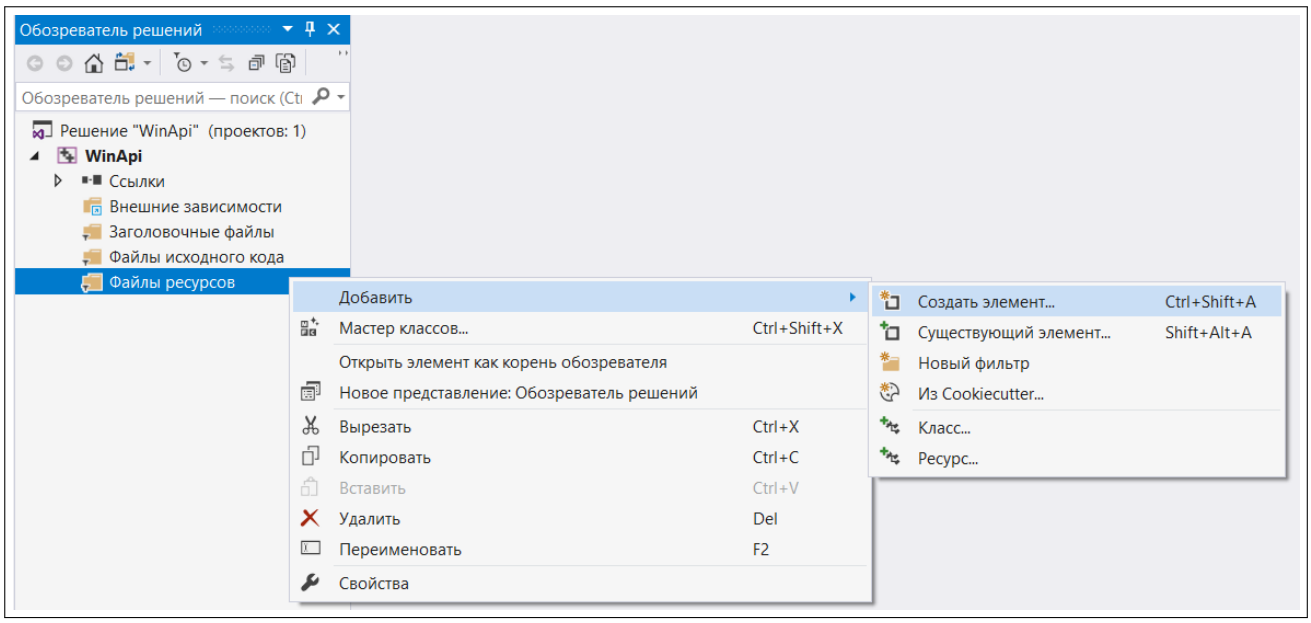



Рис. 2.2: Добавление файла messages.mc

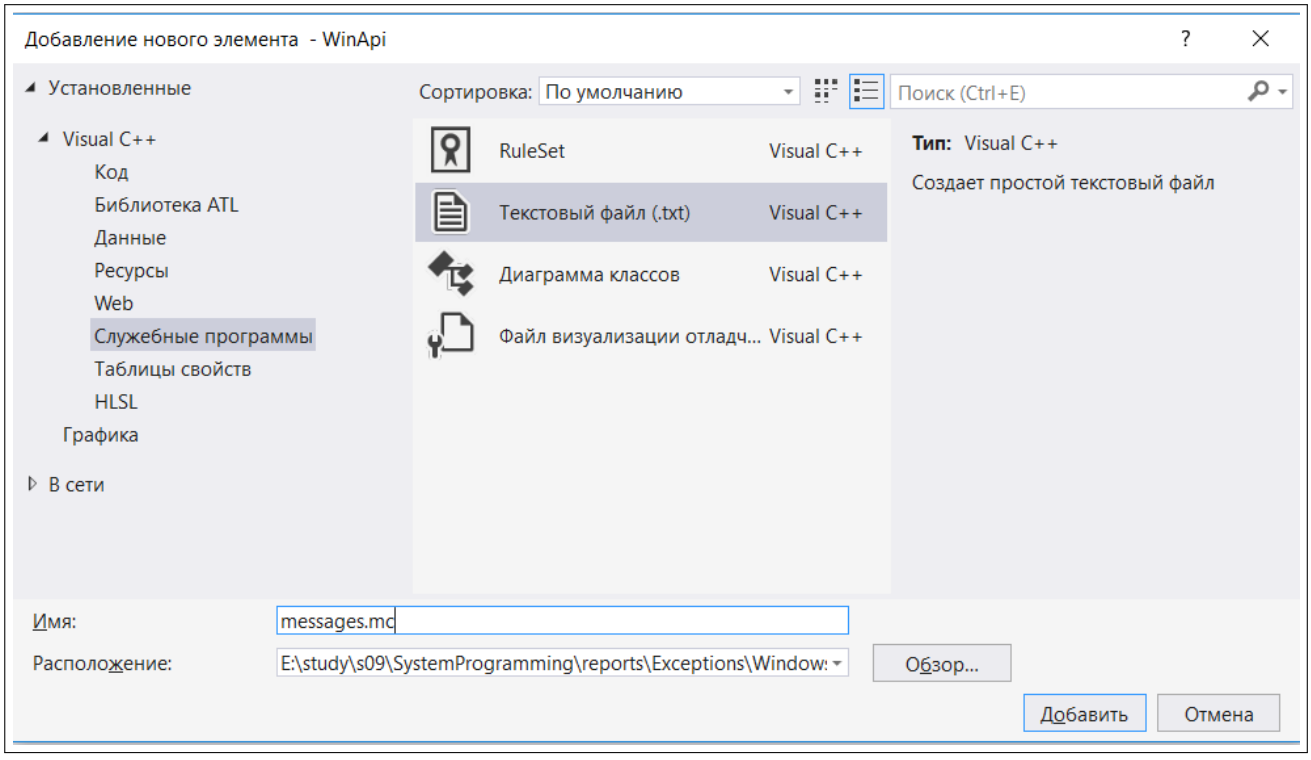



Рис. 2.3: Добавление файла messages.mc

Далее в созданный файл, необходимо добавить содержимое листинга 2.1.

```
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 .
```

Листинг 2.1: messages.mc

Содержимое файла (листинг 2.1) описывает коды для событий журнала.

Разберем его содержимое.

В начале описан язык сообщений(русский) потом две категории сообщений

(OVERFLOW_CATEGORY для событий переполнения при операции над числами с плавающей точкой; ZERODIVIDE_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

Теперь необходимо настроить среду разработки так, чтобы файл компилировался автоматически во время сборки проекта. Для этого нужно вызвать свойства файла messages.mc и в поле **Тип элемента** выбрать **Настраиваемый инструмент сборки**

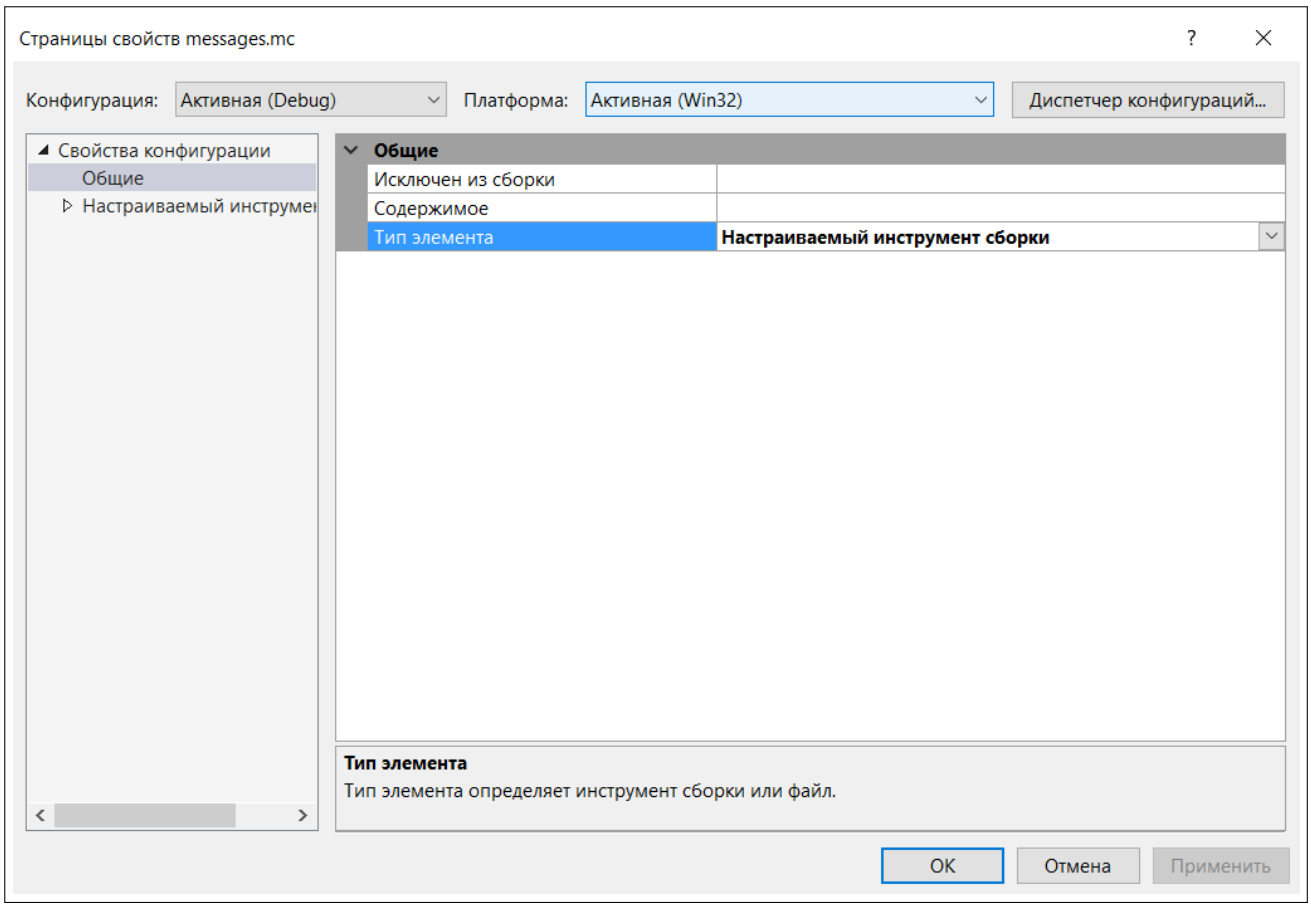



Рис. 2.4: Установка типа элемента файла messages.mc

После нажатия кнопки **Применить**, в левой панели появится вкладка **Настраиваемый инструмент сборки**, в котором необходимо ввести следующее:

**Командная строка:** mc "%(FullPath)"
**Описание:** Compiling Messages...
**Выводы:** %(Filename).rc;%(Filename).h;MSG00419.bin;MSG00409.bin

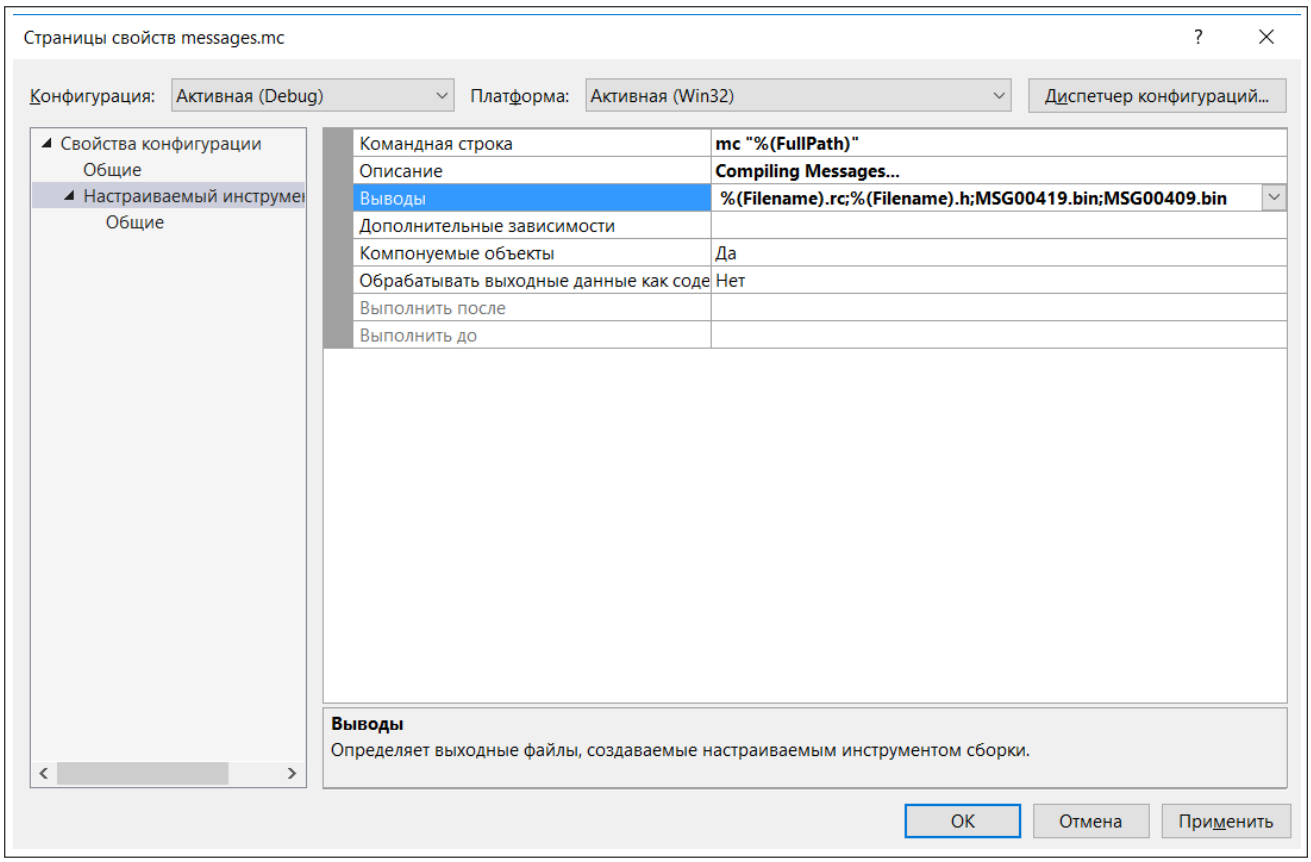



Рис. 2.5: Настройка выполнения скрипта генерации ресурсов

Теперь, при сборке проекта, ресурсы будут сгенерированы автоматически:

- message.h – заголовочный файл ресурсов (листинг 2.2). Необходимо добавить в проект.
- message.rc – файл с описанием ресурсов. Необходимо добавить в проект;
- MSG00409.bin и MSG00419.bin - бинарные файлы ресурсов.

```
1  // Language
2  // Categories
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 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)
```

Листинг 2.2: messages.h

После того, как заголовочный файл (листинг 2.2) будет добавлен в проект, можно будет пользоваться определёнными в нём константами.

После выше написанных действий, проект должен иметь структуру представленную на рисунке 2.6.

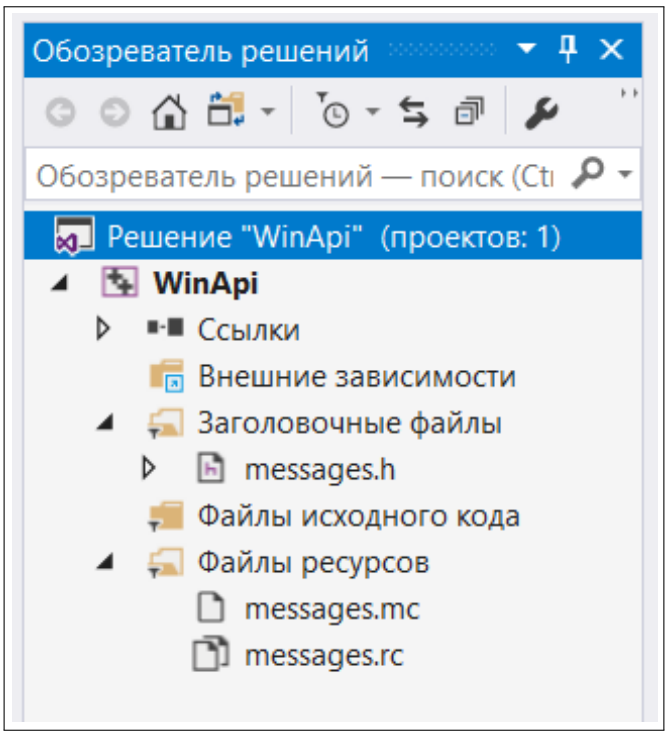



Рис. 2.6: Структура проекта WinApi

После этого с системным журналом уже можно работать, но каждое событие будет начинаться с записи:

Не удается найти описание для идентификатора события 258 из источника MyEventProvider. Вызывающий данное событие компонент не установлен на этом локальном компьютере или поврежден. Установите или восстановите компонент на локальном компьютере.

Для исправления нужно сгенерировать библиотеку с ресурсами и зарегистрировать её в системе.

Генерация библиотеки делается при помощи командной строки Visual Studio (Средства -> Командная строка Visual Studio), в котором делается переход в папку, содержащую messages.res (папка debug) и выполняется команда

**link /DLL /NOENTRY messages.res**

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\WinApi\
    ↪ WinApi\Debug>link /DLL /NOENTRY messages.res
2 Microsoft (R) Incremental Linker Version 14.11.25506.0
3 Copyright (C) Microsoft Corporation.  All rights reserved.
4
5 LINK : warning LNK4068: параметр /MACHINE не указан; принимается по умолчанию на
    ↪ X86
```

Листинг 2.3: Выполнение команды link

После выполнения этой команды будет создан файл messages.dll. Теперь необходимо зарегистрировать messages.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 2.1.

| Название | Тип | Значение | Примечание |
|---|---|---|---|
| CategoryCount | REG_DWORD | 0x00000002 | количество категорий сообщений |
| CategoryMessageFile | REG_SZ | path\messages.dll | путь до DLL |
| EventMessageFile | REG_SZ | path\messages.dll | путь до DLL |
| ParameterMessageFile | REG_SZ | path\messages.dll | путь до DLL |
| TypesSupported | REG_DWORD | 0x00000002 | количество типов сообщений |

Таблица 2.1: Значения для заполнения реестра

Для сокращения пути, файл с библиотекой стоит перенести в более подходящее место. Заполненный реестр должен выглядеть как на рисунке 2.7.

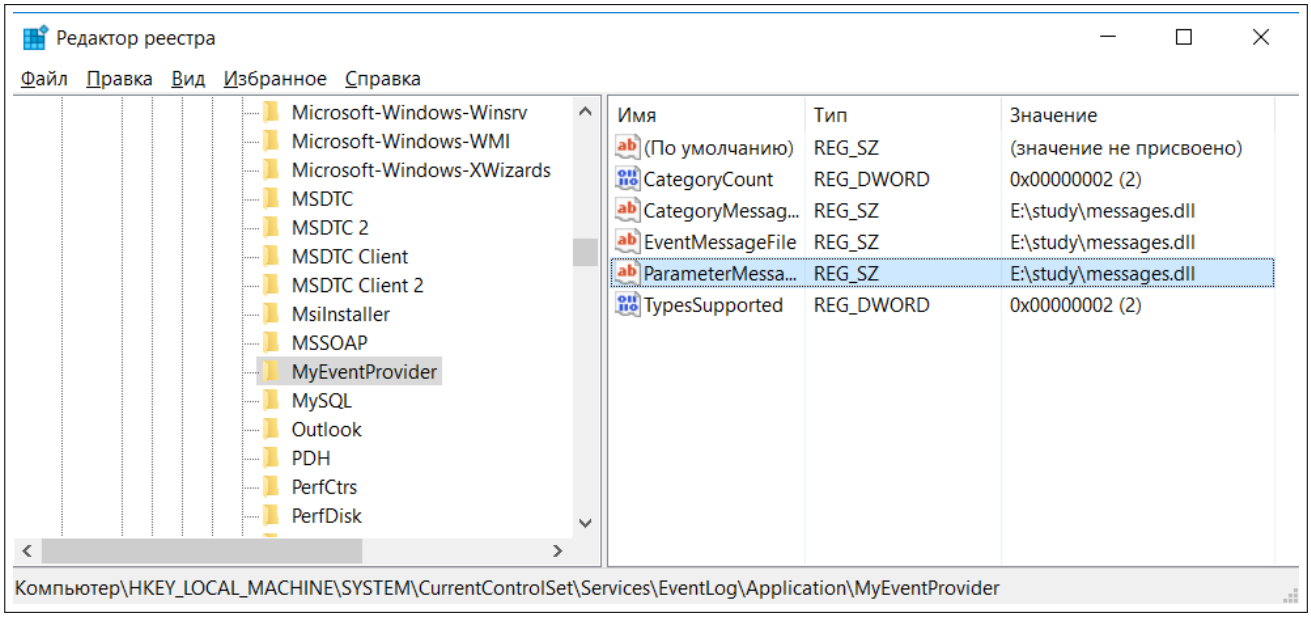



Рис. 2.7: Заполненный реестр

После этого события в системном журнале должны отображаются нормально.

## 2.4 Отладчик WinDbg

Для отладки кода и просмотра стека вызова был использован **WinDbg**.

WinDbg — позволяет отлаживать 32/64 битные приложения пользовательского уровня, драйвера, может быть использован для анализа аварийных дампов памяти, поддерживает автоматическую загрузку отладочных символов, имеется встроенный скриптовый язык для автоматизации процесса отладки, а самое главное распространяется корпорацией Microsoft совершенно бесплатно.

WinDbg при первом запуске имеет достаточно неприятный интерфейс и требуется довольно много усилий и времени для подготовки этого инструмента к комфортной работе.

Настройка интерфейса была выполнена на основе статьи [1].

Основы при работе с WinDbg были основаны на статье [2].

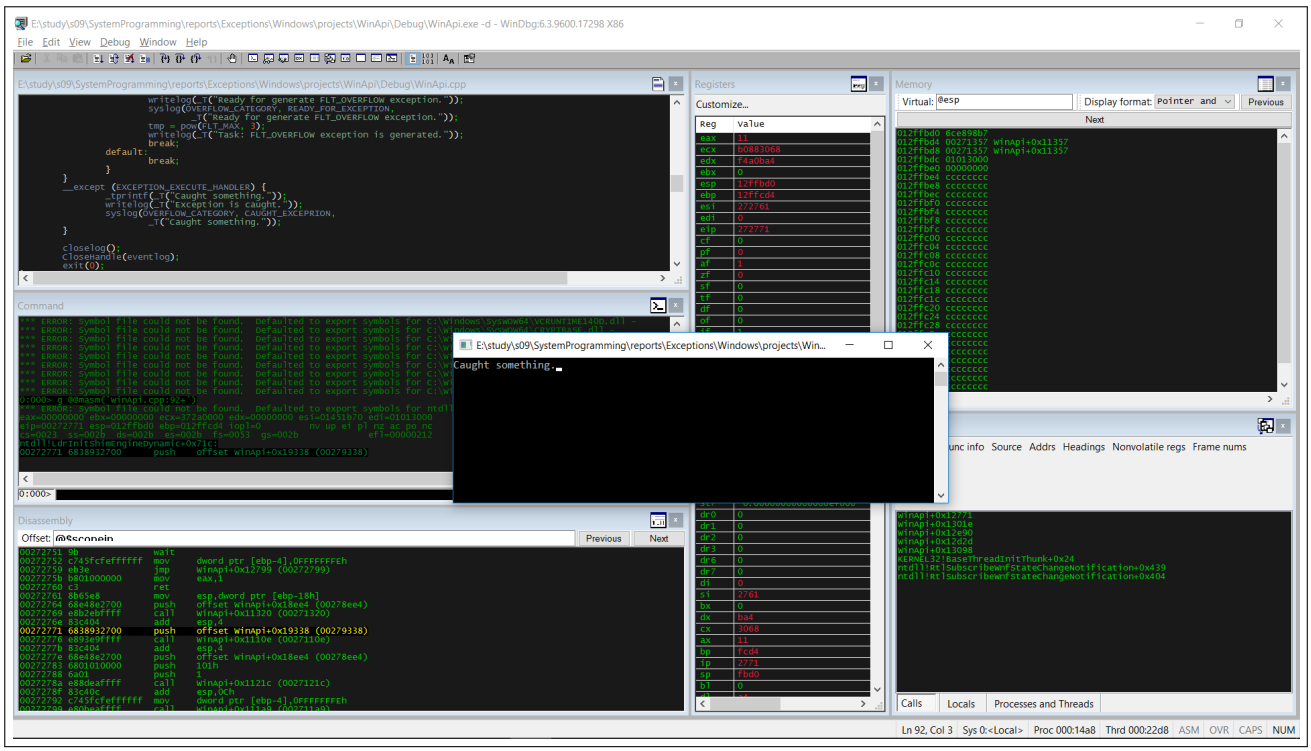


Рис. 2.8: Запуск WinApi с использованием WinDbg

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 центрального 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.

Походу работу, WinDbg оставлял желать лучшего(зависания, отсутствие подсветки текущей строки кода...) в случае чего был установлен WinDbg Preview(можно установить на windows 10 anniversary update и старше), который не умеет подобных проблем.

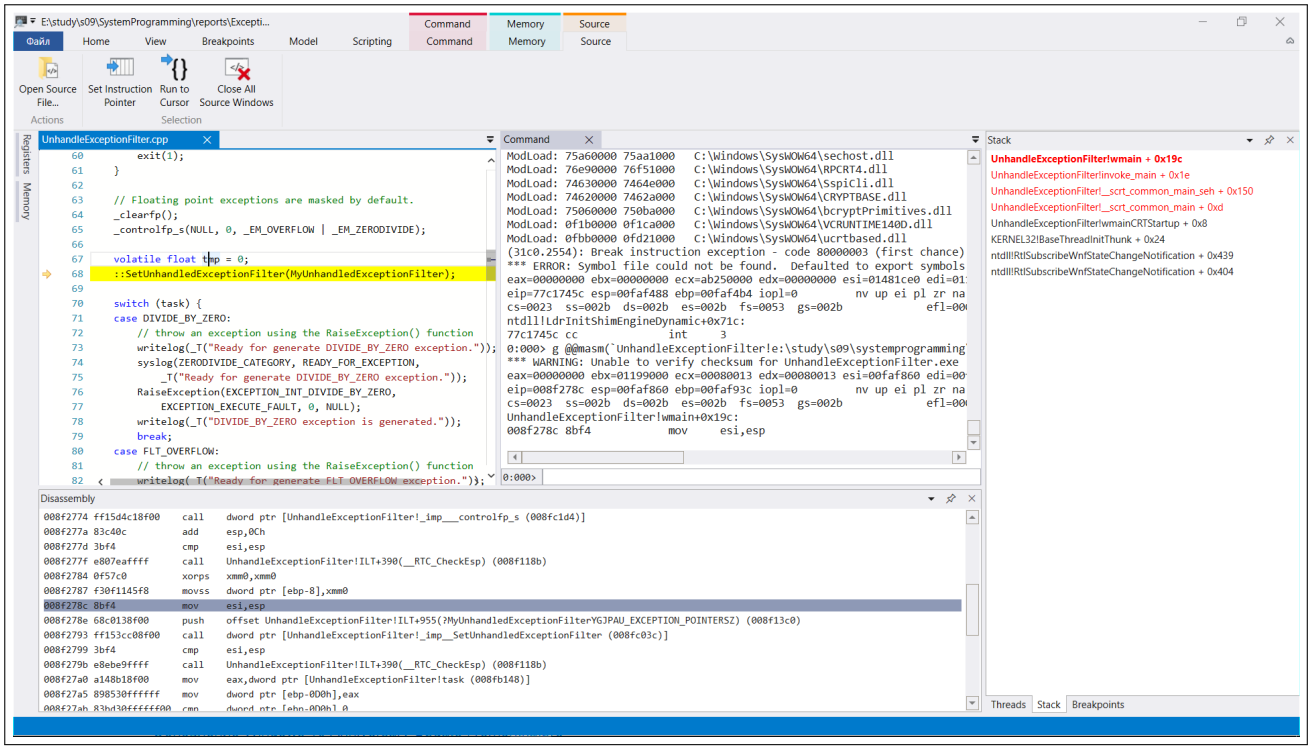


Рис. 2.9: Интерфейс WinDbg Preview

## 2.5 Сведения о системе

| Элемент | Значение |
|---|---|
| Имя ОС | Майкрософт Windows 10 Pro (Registered Trademark) |
| Версия | 10.0.14393 Сборка 14393 |
| Дополнительное описание ОС | Недоступно |
| Изготовитель ОС | Microsoft Corporation |
| Имя системы | USER-PC |
| Изготовитель | HP |
| Модель | OMEN by HP Laptop 15-ce0xx |
| Тип | Компьютер на базе x64 |
| SKU системы | 1ZB00EA#ACB |
| Процессор | Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, 2496 МГц, ядер: 4, логических процессоров: 4 |
| Версия BIOS | American Megatrends Inc. F.04, 10.05.2017 |
| Версия SMBIOS | 3.0 |
| Версия встроенного контроллера | 40.20 |
| Режим BIOS | Устаревший |
| Изготовитель основной платы | HP |
| Модель основной платы | Недоступно |

| | |
|---|---|
| Имя основной платы | Основная плата |
| Роль платформы | Мобильный |
| Состояние безопасной загрузки | Не поддерживается |
| Конфигурация PCR7 | Привязка невозможна |
| Папка Windows | C:\Windows |
| Системная папка | C:\Windows\system32 |
| Устройство загрузки | \Device\HarddiskVolume1 |
| Язык системы | Россия |
| Аппаратно-зависимый уровень (HAL) | Версия = ”10.0.14393.1378” |
| Имя пользователя | USER-PC\Tom |
| Часовой пояс | RTZ 2 (зима) |
| Установленная оперативная память (RAM) | 8,00 ГБ |
| Полный объем физической памяти | 7,87 ГБ |
| Доступно физической памяти | 3,54 ГБ |
| Всего виртуальной памяти | 12,6 ГБ |
| Доступно виртуальной памяти | 6,82 ГБ |
| Размер файла подкачки | 4,75 ГБ |
| Файл подкачки | C:\pagefile.sys |

Таблица 2.2: Информация об используемой системе

Для разработки использовалась Microsoft Visual Studio Enterprise 2017 (Версия 15.3.0).

В качестве отладчика использовался Microsoft WinDbg (release 6.3.9600.16384), работа с которым будет подробнее рассмотрена на одной из задач.

# Исключения с помощью функций WinApi

В данном разделе, с помощью функций WinApi генерируются и обрабатываются заданные в задании исключения. В листинге 3.1 показана работа с исключениями. В зависимости от параметра, передаваемого при запуске, вызывается либо исключение деления на ноль, либо переполнение разрядной сетки при работе с типом float.

```
1  #define _CRT_SECURE_NO_WARNINGS
2
3  #include <stdio.h>
4  #include <tchar.h>
5  #include <cstring>
6  #include <cfloat>
7  #include <cmath>
8  #include <excpt.h>
9  #include <windows.h>
10 #include <time.h>
11
12 #include "messages.h"
13
14 // log
15 FILE* logfile;
16 HANDLE eventlog;
17
18 void usage(const _TCHAR* prog);
19 void initlog(const _TCHAR* prog);
20 void closelog();
21 void writelog(_TCHAR* format, ...);
22 void syslog(WORD category, WORD identifier, LPWSTR message);
23
24 // Task switcher
25 enum {
26     DIVIDE_BY_ZERO,
27     FLT_OVERFLOW
28 } task;
29
30 // Defines the entry point for the console application.
31 int _tmain(int argc, _TCHAR* argv[]) {
32     //Init log
33     initlog(argv[0]);
34     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
35
36     // Check parameters number
37     if (argc != 2) {
38         _tprintf(_T("Wrong params count.\n\n"));
39         writelog(_T("Wrong params count."));
40         usage(argv[0]);
41         closelog();
42         exit(1);
43     }
44
```

```
45      // Set task
46      if (!_tcscmp(_T("-d"), argv[1])) {
47          task = DIVIDE_BY_ZERO;
48          writelog(_T("Task: DIVIDE_BY_ZERO exception."));
49      }
50      else if (!_tcscmp(_T("-o"), argv[1])) {
51          task = FLT_OVERFLOW;
52          writelog(_T("Task: FLT_OVERFLOW exception."));
53      }
54      else {
55          _tprintf(_T("Can't parse parameters.\n\n"));
56          writelog(_T("Can't parse parameters."));
57          usage(argv[0]);
58          closelog();
59          exit(1);
60      }
61
62      unsigned int newValue = _controlfp(0, 0); //получить управляющее слово,
    ↪ заданное по умолчанию
63                                                //разрешить обработку исключений с
    ↪ плавающей точкой
64      newValue &= ~(EM_OVERFLOW | EM_UNDERFLOW | EM_INEXACT | EM_ZERODIVIDE |
    ↪ EM_DENORMAL | EM_INVALID);
65      _controlfp(newValue, _MCW_EM);    //установить новое управляющее слово
66
67
68      // Set exception
69      __try {
70          volatile float tmp = 0;
71          switch (task) {
72          case DIVIDE_BY_ZERO:
73              writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
74              syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
75                  _T("Ready for generate DIVIDE_BY_ZERO exception."));
76              tmp = 1 / tmp;
77              writelog(_T("DIVIDE_BY_ZERO exception is generated."));
78              break;
79          case FLT_OVERFLOW:
80              writelog(_T("Ready for generate FLT_OVERFLOW exception."));
81              syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
82                  _T("Ready for generate FLT_OVERFLOW exception."));
83              tmp = pow(FLT_MAX, 3);
84              writelog(_T("Task: FLT_OVERFLOW exception is generated."));
85              break;
86          default:
87              break;
88          }
89      }
90      __except (EXCEPTION_EXECUTE_HANDLER) {
91          _tprintf(_T("Caught something."));
92          writelog(_T("Exception is caught."));
93          syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
94              _T("Caught something."));
95      }
96
97      closelog();
98      CloseHandle(eventlog);
99      exit(0);
100 }
101
```

```
102 // Usage manual
103 void usage(const _TCHAR* prog) {
104     _tprintf(_T("Usage: \n"));
105     _tprintf(_T("\t%s -d\n"), prog);
106     _tprintf(_T("\t\t\t for exception float divide by zero,\n"));
107     _tprintf(_T("\t%s -o\n"), prog);
108     _tprintf(_T("\t\t\t for exception float overflow.\n"));
109 }
110
111 void initlog(const _TCHAR* prog) {
112     _TCHAR logname[255];
113     wcscpy_s(logname, prog);
114
115     // replace extension
116     _TCHAR* extension;
117     extension = wcsstr(logname, _T(".exe"));
118     wcsncpy_s(extension, 5, _T(".log"), 4);
119
120     // Try to open log file for append
121     if (_wfopen_s(&logfile, logname, _T("a+"))) {
122         _wperror(_T("The following error occurred"));
123         _tprintf(_T("Can't open log file %s\n"), logname);
124         exit(1);
125     }
126
127     writelog(_T("%s is starting."), prog);
128 }
129
130 void closelog() {
131     writelog(_T("Shutting down.\n"));
132     fclose(logfile);
133 }
134
135 void writelog(_TCHAR* format, ...) {
136     _TCHAR buf[255];
137     va_list ap;
138
139     struct tm newtime;
140     __time64_t long_time;
141
142     // Get time as 64-bit integer.
143     _time64(&long_time);
144     // Convert to local time.
145     _localtime64_s(&newtime, &long_time);
146
147     // Convert to normal representation.
148     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
149         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
150         newtime.tm_min, newtime.tm_sec);
151
152     // Write date and time
153     fwprintf(logfile, _T("%s"), buf);
154     // Write all params
155     va_start(ap, format);
156     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
157     fwprintf(logfile, _T("%s"), buf);
158     va_end(ap);
159     // New sting
160     fwprintf(logfile, _T("\n"));
161 }
```

```
162
163 void syslog(WORD category, WORD identifier, LPWSTR message) {
164     LPWSTR pMessages[1] = { message };
165
166     if (!ReportEvent(
167         eventlog,                       // event log handle
168         EVENTLOG_INFORMATION_TYPE,      // event type
169         category,                       // event category
170         identifier,                     // event identifier
171         NULL,                           // user security identifier
172         1,                              // number of substitution strings
173         0,                              // data size
174         (LPCWSTR*)pMessages,            // pointer to strings
175         NULL)) {                        // pointer to binary data buffer
176         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
177     }
178 }
```

Листинг 3.1: WinApi.cpp

Стоить обратить внимание на некоторые вещи. В 70-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.

Также, в системе не удастся вызвать исключение связанные с плавающей точкой, так как они отключены. Вместо вызова исключения, система возвращает результат как NAN или INFINITY. Для включения вызова, необходимо изменить состояние слова, управляющего обработкой операций с плавающей точкой. Это возможно с помощью функции _controlfp, прототип которой выглядит следующим образом:

**unsigned int _controlfp(unsigned int new, unsigned int mask);**

Прототип определен в заголовочном файле float.h. Параметр **new** задает новое управляющее слово, а параметр **mask** является маской. Для включение вызова исключение, управляющее слова должно состоять из:

- _EM_INVALID – исключение EXCEPTION_FLT_INVALID_OPERATION;
- _EM_DENORMAL – исключение EXCEPTION_FLT_DENORMAL_OPERAND;
- _EM_ZERODIVIDE – исключение EXCEPTION_FLT_DIVIDE_BY_ZERO;
- _EM_OVERFLOW – исключение EXCEPTION_FLT_OVERFLOW;
- _EM_UNDERFLOW – исключение EXCEPTION_FLT_UNDERFLOW;
- _EM_INEXACT – исключение EXCEPTION_FLT_INEXACT_RESULT.

Маска в данном случае, должна принимать значение **_MCW_EM** или **0xffffffff**.

В данной задаче мы не разбираем, какое именно исключение произошло, поэтому в системный журнал все пойманные события помечаются как события из группы переполнения.

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\WinApi\Debug
    ↪ >WinApi.exe
2 Wrong params count.
3
```

```
Usage:
        WinApi.exe -d
                        for exception float divide by zero,
        WinApi.exe -o
                        for exception float overflow.

E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\WinApi\Debug
    ↪ >WinApi.exe -d
Caught something.
E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\WinApi\Debug
    ↪ >WinApi.exe -o
Caught something.
```

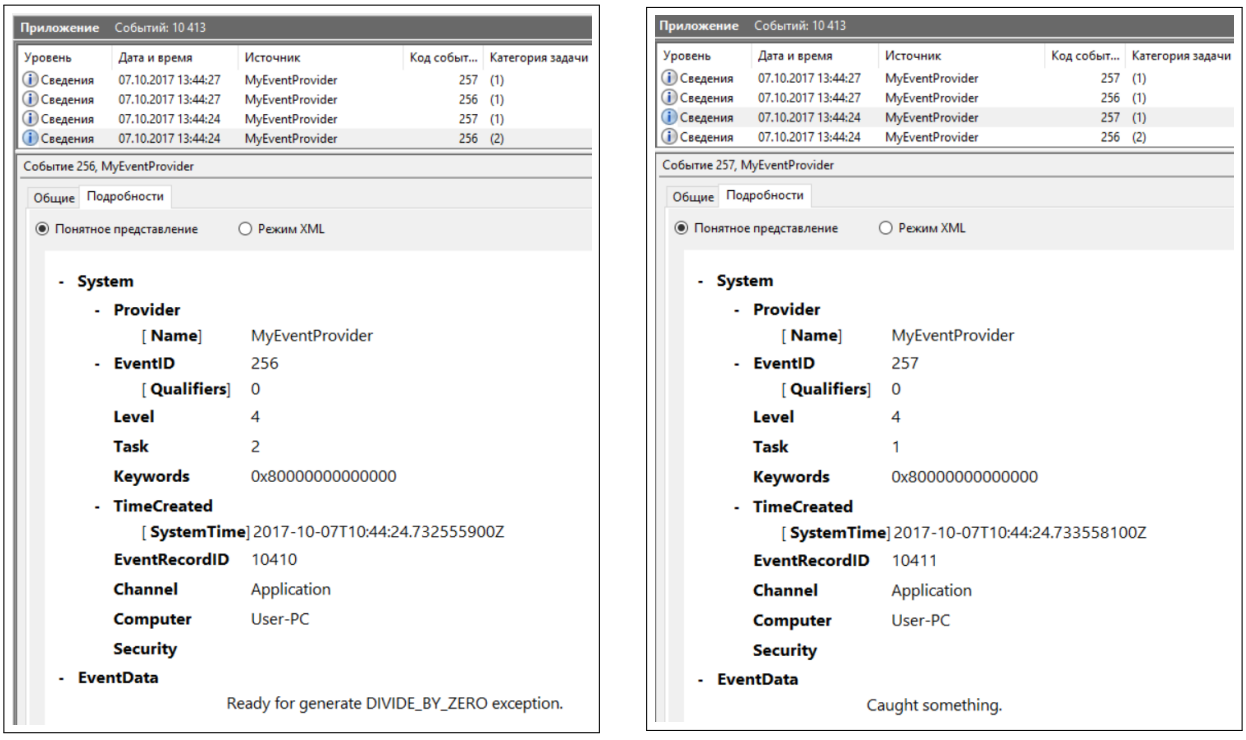
Листинг 3.2: Лог консоли при запуске WinApi.exe

```
[7/10/2017 13:44:22] WinApi.exe is starting.
[7/10/2017 13:44:22] Wrong params count.
[7/10/2017 13:44:22] Shutting down.

[7/10/2017 13:44:24] WinApi.exe is starting.
[7/10/2017 13:44:24] Task: DIVIDE_BY_ZERO exception.
[7/10/2017 13:44:24] Ready for generate DIVIDE_BY_ZERO exception.
[7/10/2017 13:44:24] Exception is caught.
[7/10/2017 13:44:24] Shutting down.

[7/10/2017 13:44:27] WinApi.exe is starting.
[7/10/2017 13:44:27] Task: FLT_OVERFLOW exception.
[7/10/2017 13:44:27] Ready for generate FLT_OVERFLOW exception.
[7/10/2017 13:44:27] Exception is caught.
[7/10/2017 13:44:27] Shutting down.
```

Листинг 3.3: Содержимое файла WinApi.log




(a) Событие перед генерацией исключения (b) Событие о произошедшем исключении

Рис. 3.1: Просмотр событий

Чего и требовалось ожидать, событие было зафиксировано в журнале событий, но в нем меньше полезной информации чем в лог-файле, далее все логи будут сохранятся в папку logs, но в отчете приводиться не будут.

Большая часть кода из листинга 3.1 будет использовать и дальше в работе, поэтому рассмотрим листинг подробнее. В программе имеется 5 функций:

1. usage - функция информативного характера, уведомляет пользователя как пользоваться программой;
2. initlog - инициализация лога(создание лог файла рядом с исполняемой программой);
3. closelog - закрытие записи в лог-файл:
4. writelog - добавление новых записей в лог-файл;
5. syslog - регистрация события в журнале событий Windows.

Рассмотрим функцию syslog более подробно. В ней происходит вызов команды **ReportEvent**, которая непосредственно передаёт строку в системный журнал. Синтаксис команды следующий:

```
1  BOOL ReportEvent(
2  _In_ HANDLE hEventLog,
3  _In_ WORD wType,
4  _In_ WORD wCategory,
5  _In_ DWORD dwEventID,
6  _In_ PSID lpUserSid,
7  _In_ WORD wNumStrings,
8  _In_ DWORD dwDataSize,
9  _In_ LPCTSTR *lpStrings,
10 _In_ LPVOID lpRawData
11 );
```

Листинг 3.4: Синтаксис команды ReportEvent

Значения полей следующие:

- hEventLog – описатель логера (инициализирован в функции initlog);
- 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);
- 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.

Теперь можно перейти к отладке кода при помощи WinDbg.

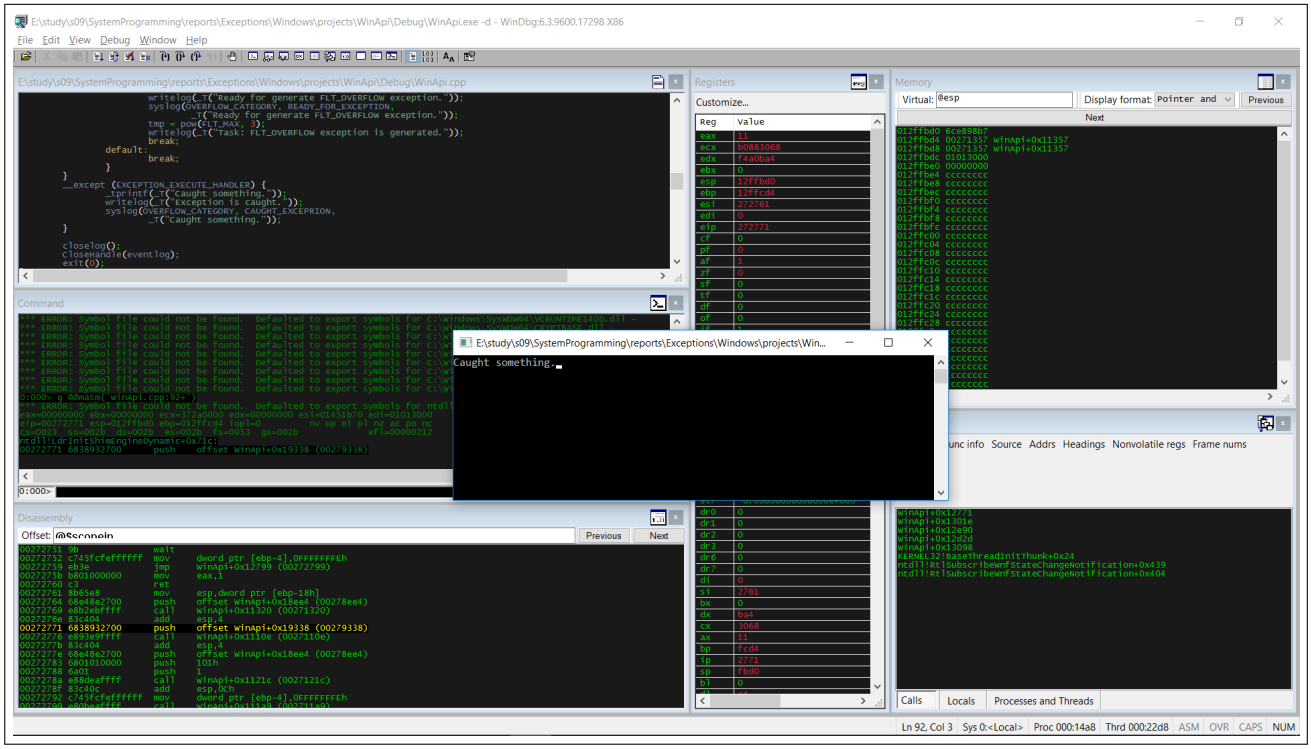



Рис. 3.2: Запуск WinApi с использованием WinDbg

В правом нижнем углу показан стек. В данном случае глубина стека не достаточно большая для наглядного изучения поиска обработчика, но вызов обработчика на нём виден.

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

# Использование GetExceptionCode

Функция GetExceptionCode позволяет получить код произошедшего исключения. Вызов функции возможен только в выражении-фильтре или в блоке обработки исключения.

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <tchar.h>
#include <cstring>
#include <cfloat>
#include <cmath>
#include <excpt.h>
#include <windows.h>
#include <time.h>

#include "messages.h"

// log
FILE* logfile;
HANDLE eventlog;

void usage(const _TCHAR* prog);
void initlog(const _TCHAR* prog);
void closelog();
void writelog(_TCHAR* format, ...);
void syslog(WORD category, WORD identifier, LPWSTR message);

// Task switcher
enum {
    DIVIDE_BY_ZERO,
    FLT_OVERFLOW
} task;


// Defines the entry point for the console application.
int _tmain(int argc, _TCHAR* argv[]) {
    //Init log
    initlog(argv[0]);
    eventlog = RegisterEventSource(NULL, L"MyEventProvider");

    // Check parameters number
    if (argc != 2) {
        _tprintf(_T("Wrong params count.\n\n"));
        writelog(_T("Wrong params count."));
        usage(argv[0]);
        closelog();
        exit(1);
    }

```

```
46      // Set task
47      if (!_tcscmp(_T("-d"), argv[1])) {
48          task = DIVIDE_BY_ZERO;
49          writelog(_T("Task: DIVIDE_BY_ZERO exception."));
50      }
51      else if (!_tcscmp(_T("-o"), argv[1])) {
52          task = FLT_OVERFLOW;
53          writelog(_T("Task: FLT_OVERFLOW exception."));
54      }
55      else {
56          _tprintf(_T("Can't parse parameters.\n\n"));
57          writelog(_T("Can't parse parameters."));
58          usage(argv[0]);
59          closelog();
60          exit(1);
61      }
62
63      unsigned int newValue = _controlfp(0, 0); //получить управляющее слово,
    ↪ заданное по умолчанию
64                                                //разрешить обработку исключений с
    ↪ плавающей точкой
65      newValue &= ~(EM_OVERFLOW | EM_UNDERFLOW | EM_INEXACT | EM_ZERODIVIDE |
    ↪ EM_DENORMAL | EM_INVALID);
66      _controlfp(newValue, _MCW_EM);    //установить новое управляющее слово
67
68      // Set exception
69      int a = 1;
70      int b = 0;
71      volatile float tmp = 0;
72      switch (task) {
73      case DIVIDE_BY_ZERO:
74          writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
75          syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
76              _T("Ready for generate DIVIDE_BY_ZERO exception."));
77          __try {
78              tmp = a / b;
79          }
80          __except(GetExceptionCode() == EXCEPTION_INT_DIVIDE_BY_ZERO ?
81              EXCEPTION_EXECUTE_HANDLER : EXCEPTION_CONTINUE_SEARCH) {
82              writelog(_T(" Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO "))
    ↪ ;
83              _tprintf(_T(" Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO "))
    ↪ ;
84              syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
85                  _T(" Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO ."));
86          }
87
88          writelog(_T("DIVIDE_BY_ZERO exception is generated."));
89          break;
90      case FLT_OVERFLOW:
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          __try {
95              tmp = 1 / pow(99999, 999999);
96          }
97          __except (EXCEPTION_EXECUTE_HANDLER) {
98              DWORD exceptionCode = GetExceptionCode();
99              if (exceptionCode == EXCEPTION_FLT_OVERFLOW) {
100                 writelog(_T(" Caught exception is: EXCEPTION_FLT_OVERFLOW "));
```

```
101                 _tprintf(_T(" Caught exception is: EXCEPTION_FLT_OVERFLOW "));
102                 syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
103                     _T(" Caught exception is: EXCEPTION_FLT_OVERFLOW ."));
104             }
105             else {
106                 writelog(_T(" UNKNOWN exception : %x\n"), exceptionCode);
107                 _tprintf(_T(" UNKNOWN exception : %x\n"), exceptionCode);
108             }
109         }
110         writelog(_T("Task: FLT_OVERFLOW exception is generated."));
111         break;
112     default:
113         break;
114     }
115
116
117
118     closelog();
119     CloseHandle(eventlog);
120     exit(0);
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     wcscpy_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         _wperror(_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 }
183
184 void syslog(WORD category, WORD identifier, LPWSTR message) {
185     LPWSTR pMessages[1] = { message };
186
187     if (!ReportEvent(
188         eventlog,                   // event log handle
189         EVENTLOG_INFORMATION_TYPE,  // event type
190         category,                   // event category
191         identifier,                 // event identifier
192         NULL,                       // user security identifier
193         1,                          // number of substitution strings
194         0,                          // data size
195         (LPCWSTR*)pMessages,        // pointer to strings
196         NULL)) {                    // pointer to binary data buffer
197         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
198     }
199 }
```

Листинг 4.1: GetException.cpp

Используя функция GetExceptionCode, исключения были успешны определены и обработаны непосредственно в обработчике. Запуск под отладчиком показывает практически аналогичный предыдущему результат, различия имеются в логе консоли и логфайле.

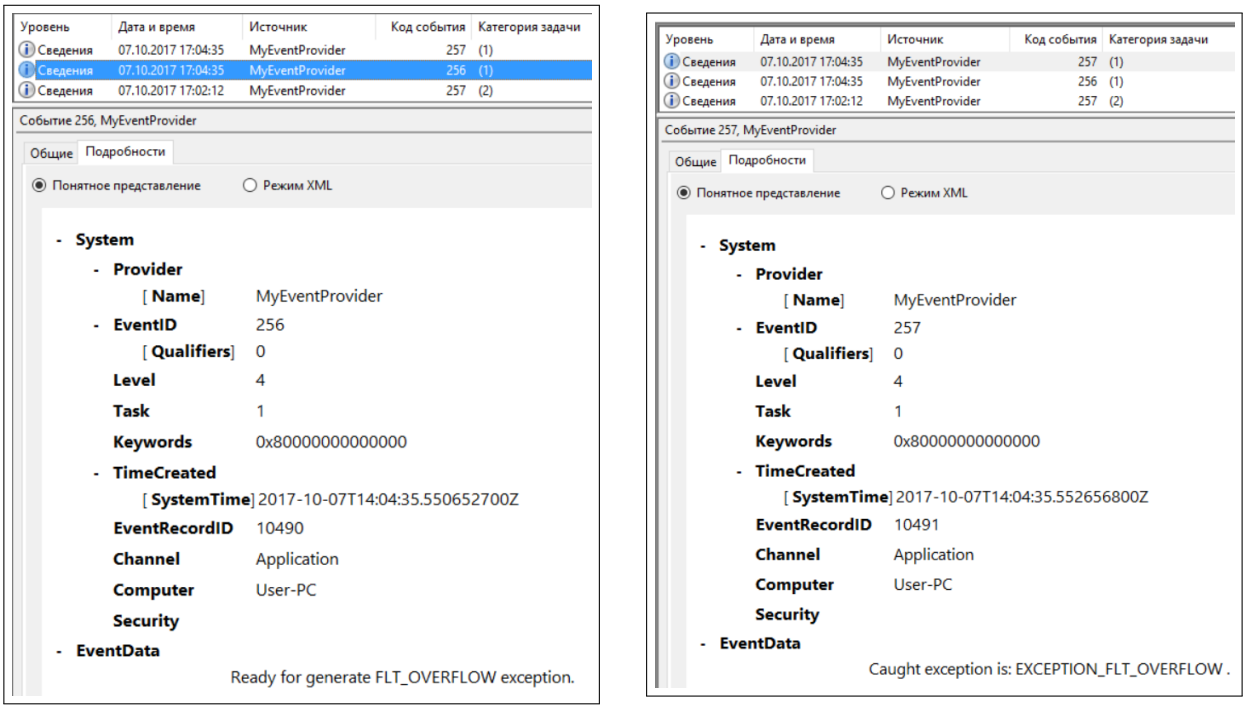
```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
    ↪ GetExceptionCode\Debug>GetExceptionCode.exe -d
2  Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO
3 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
    ↪ GetExceptionCode\Debug>GetExceptionCode.exe -o
4  Caught exception is: EXCEPTION_FLT_OVERFLOW
```

Листинг 4.2: Лог консоли при запуске GetExceptionCode.exe

```
1 [7/10/2017 17:2:12] GetExceptionCode.exe is starting.
2 [7/10/2017 17:2:12] Task: DIVIDE_BY_ZERO exception.
3 [7/10/2017 17:2:12] Ready for generate DIVIDE_BY_ZERO exception.
4 [7/10/2017 17:2:12]  Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO
5 [7/10/2017 17:2:12] Shutting down.
```

```
6
7  [7/10/2017 17:4:35] GetExceptionCode.exe is starting.
8  [7/10/2017 17:4:35] Task: FLT_OVERFLOW exception.
9  [7/10/2017 17:4:35] Ready for generate FLT_OVERFLOW exception.
10 [7/10/2017 17:4:35]  Caught exception is: EXCEPTION_FLT_OVERFLOW
11 [7/10/2017 17:4:35] Shutting down.
```

Листинг 4.3: Содержимое файла GetExceptionCode.log




(a) Событие перед генерацией исключения

(b) Событие о произошедшем исключении

Рис. 4.1: Просмотр событий

В журнале событий сгенерированное исключение, также успешно определено.

# Собственная функция фильтр

Для более детальной обработки исключения, используются функция-фильтр. Такие функции как GetExceptionCode и GetExceptionInformation, в случае необходимости следует передавать как параметры функции фильтра. В нижеприведенной программе, как парметр функции фильтра передается код ошибки.

```
1  #define _CRT_SECURE_NO_WARNINGS
2
3  #include <stdio.h>
4  #include <tchar.h>
5  #include <cstring>
6  #include <cfloat>
7  #include <cmath>
8  #include <excpt.h>
9  #include <windows.h>
10 #include <time.h>
11
12 #include "messages.h"
13
14 // log
15 FILE* logfile;
16 HANDLE eventlog;
17
18 void usage(const _TCHAR* prog);
19 void initlog(const _TCHAR* prog);
20 void closelog();
21 void writelog(_TCHAR* format, ...);
22 LONG Filter(DWORD dwExceptionGode);
23 void syslog(WORD category, WORD identifier, LPWSTR message);
24
25 // Task switcher
26 enum {
27     DIVIDE_BY_ZERO,
28     FLT_OVERFLOW
29 } task;
30
31 // Defines the entry point for the console application.
32 int _tmain(int argc, _TCHAR* argv[]) {
33     //Init log
34     initlog(argv[0]);
35     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
36
37     // Check parameters number
38     if (argc != 2) {
39         _tprintf(_T("Too few parameters.\n\n"));
40         writelog(_T("Too few parameters."));
41         usage(argv[0]);
42         closelog();
43         exit(1);
44     }
```

```
45
46      // Set task
47      if (!_tcscmp(_T("-d"), argv[1])) {
48          task = DIVIDE_BY_ZERO;
49          writelog(_T("Task: DIVIDE_BY_ZERO exception."));
50      }
51      else if (!_tcscmp(_T("-o"), argv[1])) {
52          task = FLT_OVERFLOW;
53          writelog(_T("Task: FLT_OVERFLOW exception."));
54      }
55      else {
56          _tprintf(_T("Can't parse parameters.\n\n"));
57          writelog(_T("Can't parse parameters."));
58          usage(argv[0]);
59          closelog();
60          exit(1);
61      }
62
63      unsigned int newValue = _controlfp(0, 0); //получить управляющее слово,
    ↪ заданное по умолчанию
64                                                //разрешить обработку исключений с
    ↪ плавающей точкой
65      newValue &= ~(EM_OVERFLOW | EM_UNDERFLOW | EM_INEXACT | EM_ZERODIVIDE |
    ↪ EM_DENORMAL | EM_INVALID);
66      _controlfp(newValue, _MCW_EM);    //установить новое управляющее слово
67
68      // Set exception
69      __try {
70          int a = 1;
71          int b = 0;
72          volatile float tmp = 0;
73          switch (task) {
74          case DIVIDE_BY_ZERO:
75              writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
76              syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
77                  _T("Ready for generate DIVIDE_BY_ZERO exception."));
78              tmp = a / b;
79              writelog(_T("DIVIDE_BY_ZERO exception is generated."));
80              break;
81          case FLT_OVERFLOW:
82              writelog(_T("Ready for generate FLT_OVERFLOW exception."));
83              syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
84                  _T("Ready for generate FLT_OVERFLOW exception."));
85              tmp = 1 / pow(99999, 999999);
86              writelog(_T("Task: FLT_OVERFLOW exception is generated."));
87              break;
88          default:
89              break;
90          }
91      }
92      // Own filter function.
93      __except (Filter(GetExceptionCode())) {
94          switch (GetExceptionCode()) {
95          case EXCEPTION_INT_DIVIDE_BY_ZERO:
96              _tprintf(_T("Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO"));
97              writelog(_T("Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO"));
98              syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
99                  _T("Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO."));
100             break;
101         case EXCEPTION_FLT_OVERFLOW:
```

```
102             _tprintf(_T("Caught exception is: EXCEPTION_FLT_OVERFLOW"));
103             writelog(_T("Caught exception is: EXCEPTION_FLT_OVERFLOW"));
104             syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
105                 _T("Caught exception is: EXCEPTION_FLT_OVERFLOW."));
106             break;
107         default:
108             _tprintf(_T("UNKNOWN exception: %x\n"), GetExceptionCode());
109             writelog(_T("UNKNOWN exception: %x\n"), GetExceptionCode());
110         }
111     }
112     closelog();
113     CloseHandle(eventlog);
114     exit(0);
115 }
116
117 // Own filter function.
118 LONG Filter(DWORD dwExceptionGode) {
119     _tprintf(_T("Filter function used\n"));
120     if (dwExceptionGode == EXCEPTION_INT_DIVIDE_BY_ZERO || dwExceptionGode ==
    ↪ EXCEPTION_FLT_OVERFLOW)
121         return EXCEPTION_EXECUTE_HANDLER;
122     return EXCEPTION_CONTINUE_SEARCH;
123 }
124
125 // Usage manual
126 void usage(const _TCHAR* prog) {
127     _tprintf(_T("Usage: \n"));
128     _tprintf(_T("\t%s -d\n"), prog);
129     _tprintf(_T("\t\t\t for exception float divide by zero,\n"));
130     _tprintf(_T("\t%s -o\n"), prog);
131     _tprintf(_T("\t\t\t for exception float overflow.\n"));
132 }
133
134 void initlog(const _TCHAR* prog) {
135     _TCHAR logname[255];
136     wcscpy_s(logname, prog);
137
138     // replace extension
139     _TCHAR* extension;
140     extension = wcsstr(logname, _T(".exe"));
141     wcsncpy_s(extension, 5, _T(".log"), 4);
142
143     // Try to open log file for append
144     if (_wfopen_s(&logfile, logname, _T("a+"))) {
145         _wperror(_T("The following error occurred"));
146         _tprintf(_T("Can't open log file %s\n"), logname);
147         exit(1);
148     }
149
150     writelog(_T("%s is starting."), prog);
151 }
152
153 void closelog() {
154     writelog(_T("Shutting down.\n"));
155     fclose(logfile);
156 }
157
158 void writelog(_TCHAR* format, ...) {
159     _TCHAR buf[255];
160     va_list ap;
```

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

Листинг 5.1: FilterFunction.cpp

Исключения успешно обработались в функции-фильтре.

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
    ↪ FilterFunction\Debug>FilterFunction.exe -d
2 Filter function used
3 Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO
4 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
    ↪ FilterFunction\Debug>FilterFunction.exe -o
5 Filter function used
6 Caught exception is: EXCEPTION_FLT_OVERFLOW
```

Листинг 5.2: Лог консоли при запуске FilterException.exe

```
1 [7/10/2017 18:2:28] FilterFunction.exe is starting.
2 [7/10/2017 18:2:28] Task: DIVIDE_BY_ZERO exception.
3 [7/10/2017 18:2:28] Ready for generate DIVIDE_BY_ZERO exception.
4 [7/10/2017 18:2:28] Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO
```

```
5  [7/10/2017 18:2:28] Shutting down.
6
7  [7/10/2017 18:2:30] FilterFunction.exe is starting.
8  [7/10/2017 18:2:30] Task: FLT_OVERFLOW exception.
9  [7/10/2017 18:2:30] Ready for generate FLT_OVERFLOW exception.
10 [7/10/2017 18:2:30] Caught exception is: EXCEPTION_FLT_OVERFLOW
11 [7/10/2017 18:2:30] Shutting down.
```

Листинг 5.3: Содержимое файла FilterException.log

При отладке программы, эксперементальным путем было выявлено что управление не передается в то место, где определена функция-фильтр, видимо путем оптимизации исходного кода, она подставляется в место вызова.

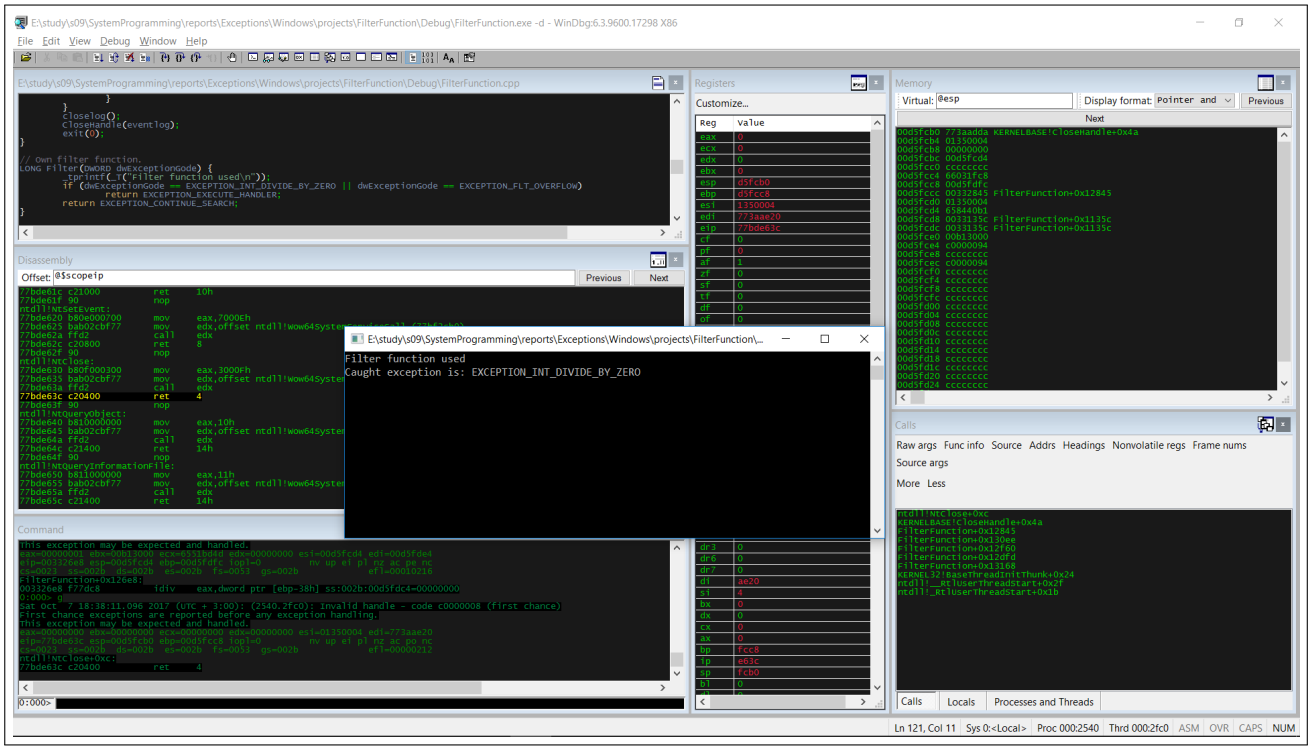



Рис. 5.1: Запуск FilterFunction с использованием WinDbg

# Использование функций RaiseException и GetExceptionInformation

Исключение можно возбудить не только в результате каких-то арифметических или ло- гических операций, но и искусственным образом, вызвав функцию RaiseException, она имеет следующий прототип.

```
1 VOID RaiseException(
2     DWORD dwExceptionCode,
3     DWORD dwExceptionFlags,
4     DWORD nNumberOfArguments,
5     CONST ULONG_PTR *pArguments);
```

Листинг 6.1: Прототип функции RaiseException

Наиболее важным параметром является первый, который определяет тип возбуждаемого исключения

Более подробную информацию об исключении можно получить при помощи вызова функции **GetExceptionInformation**, которая имеет следующий прототип:

**LPEXCEPTION_POINTERS GetExceptionInformation(VOID);**

Функция возвращает указатель на структуру следующего типа:

```
1 struct _EXCEPTION_POINTERS {
2     EXCEPTION_RECORD *ExceptionRecord,
3         CONTEXT *ContextRecord
4 }
```

Листинг 6.2: Структура _EXCEPTION_POINTERS

В структуру типа ContextRecord система записывает содержимое всех регистров процессора на момент исключения. Эта структура имеет описание, которое можно найти в заголовочном файле WinNt.h.

В структуру типа ExceptionRecord записывается подробная машинно-независимая информация о последнем исключении.

```
1  typedef struct _EXCEPTION_RECORD
2  {
3      DWORD ExceptionCode;
4      DWORD ExcepLionFlags;
5      struct _EXCEPTION_RECORD *ExceptionRecord;
6      PVOID ExceptionAddress;
7      DWORD NurrberParameters;
8      ULONG_PTR ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
9
10 } EXCEPTION_RECORD;
```

Листинг 6.3: Структура _EXCEPTION_RECORD

Разберем элементы структуры:

- ExceptionCode — код исключения. Это информация, возвращаемая функцией GetExceptionCode.
- ExceptionFlags — флаги исключения. На данный момент определено только два значения: 0 (возобновляемое исключение) и EXCEPTION_NONCONTINUABLE (не возобновляемое исключение). Любая попытка возобновить работу программ мы после не возобновляемого исключения генерирует исключение EXCEPTION_NONCONTINUABLE_EXCEPTION.
- ExceptionRecord - указатель на структуру EXCEPTION_RECORD, содержащую информацию о другом необработанном исключении при обработке одного исключения может возникнуть другое. Например, код внутри фильтра исключений может попытаться выполнить деление на нуль. Когда возникает серия вложенных исключений, записи с информацией о них могут образовывать связанный список Исключение будет вложенным, если оно генерируется при обработке фильтра. В отсутствие необработанных исключений ExceptionRecord равен NULL.
- ExceptionAddress — адрес машинной команды, при выполнении которой про изошло исключение
- NumberParameters — количество параметров, связанных с исключением (0-15). Это число заполненных элементов в массиве ExceptionInformation. Почти для всех исключений значение этого элемента равно 0.
- ExceptionInformation — массив дополнительных аргументов, описывающих ис ключение. Почти для всех исключений элементы этого массива не определены.

Далее приведены программы, которые выводят на консоль информацию об исключении, используя для получения этой информации функцию GetExceptionInformation.

```
1  #include <stdio.h>
2  #include <tchar.h>
3  #include <cstring>
4  #include <cfloat>
5  #include <cmath>
6  #include <excpt.h>
7  #include <windows.h>
8  #include <time.h>
9
10 #include "messages.h"
11
12 // log
13 FILE* logfile;
14 HANDLE eventlog;
15
16 void usage(const _TCHAR* prog);
17 void initlog(const _TCHAR* prog);
18 void closelog();
19 void writelog(_TCHAR* format, ...);
20 LONG Filter(DWORD dwExceptionGode, const _EXCEPTION_POINTERS *ep);
21 void syslog(WORD category, WORD identifier, LPWSTR message);
22
23 // Task switcher
24 enum {
25     DIVIDE_BY_ZERO,
```

```
26      FLT_OVERFLOW
27  } task;
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      // Check parameters number
36      if (argc != 2) {
37          _tprintf(_T("Too few parameters.\n\n"));
38          writelog(_T("Too few parameters."));
39          usage(argv[0]);
40          closelog();
41          exit(1);
42      }
43
44      // Set task
45      if (!_tcscmp(_T("-d"), argv[1])) {
46          task = DIVIDE_BY_ZERO;
47          writelog(_T("Task: DIVIDE_BY_ZERO exception."));
48      }
49      else if (!_tcscmp(_T("-o"), argv[1])) {
50          task = FLT_OVERFLOW;
51          writelog(_T("Task: FLT_OVERFLOW exception."));
52      }
53      else {
54          _tprintf(_T("Can't parse parameters.\n\n"));
55          writelog(_T("Can't parse parameters."));
56          usage(argv[0]);
57          closelog();
58          exit(1);
59      }
60
61      __try {
62          switch (task) {
63          case DIVIDE_BY_ZERO:
64              // throw an exception using the RaiseException() function
65              writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
66              syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
67                  _T("Ready for generate DIVIDE_BY_ZERO exception."));
68              RaiseException(EXCEPTION_INT_DIVIDE_BY_ZERO, 0, 0, NULL);
69              writelog(_T("DIVIDE_BY_ZERO exception is generated."));
70              break;
71          case FLT_OVERFLOW:
72              // throw an exception using the RaiseException() function
73              writelog(_T("Ready for generate FLT_OVERFLOW exception."));
74              syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
75                  _T("Ready for generate FLT_OVERFLOW exception."));
76              RaiseException(EXCEPTION_FLT_OVERFLOW, 0, 0, NULL);
77              writelog(_T("Task: FLT_OVERFLOW exception is generated."));
78              break;
79          default:
80              break;
81          }
82      }
83      __except (Filter(GetExceptionCode(), GetExceptionInformation())) {
84          // There is nothing to do, everything is done in the filter function.
85      }
```

```
86      closelog();
87      CloseHandle(eventlog);
88      exit(0);
89  }
90
91  LONG Filter(DWORD dwExceptionGode, const _EXCEPTION_POINTERS *ExceptionPointers
        ↪ ) {
92
93      _TCHAR buf[400] = { '\0' };
94      const _TCHAR* err = _T("Fatal error!\nexeption code: 0x");
95      const _TCHAR* mes = _T("\nProgram terminate!");
96      if (ExceptionPointers) {
97          EXCEPTION_RECORD er = *ExceptionPointers->ExceptionRecord;
98          swprintf_s(buf, _T("%s%x%s%x%s%x%s%x"),
99              _T("ExceptionCode: "), er.ExceptionCode,
100             _T(", ExceptionFlags: "), er.ExceptionFlags,
101             _T(", ExceptionRecord: "), er.ExceptionRecord,
102             _T(", ExceptionAddress: "), er.ExceptionAddress,
103             _T(", NumberParameters: "), er.NumberParameters);
104     }
105     else
106         swprintf_s(buf, _T("%s%x%s"), err, dwExceptionGode, mes);
107
108     _tprintf(_T("%s"), buf);
109     writelog(_T("%s"), buf);
110     syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION, buf);
111
112     return EXCEPTION_EXECUTE_HANDLER;
113 }
114
115 // Usage manual
116 void usage(const _TCHAR* prog) {
117     _tprintf(_T("Usage: \n"));
118     _tprintf(_T("\t%s -d\n"), prog);
119     _tprintf(_T("\t\t\t for exception float divide by zero,\n"));
120     _tprintf(_T("\t%s -o\n"), prog);
121     _tprintf(_T("\t\t\t for exception float overflow.\n"));
122 }
123
124 void initlog(const _TCHAR* prog) {
125     _TCHAR logname[255];
126     wcscpy_s(logname, prog);
127
128     // replace extension
129     _TCHAR* extension;
130     extension = wcsstr(logname, _T(".exe"));
131     wcsncpy_s(extension, 5, _T(".log"), 4);
132
133     // Try to open log file for append
134     if (_wfopen_s(&logfile, logname, _T("a+"))) {
135         _wperror(_T("The following error occurred"));
136         _tprintf(_T("Can't open log file %s\n"), logname);
137         exit(1);
138     }
139
140     writelog(_T("%s is starting."), prog);
141 }
142
143 void closelog() {
144     writelog(_T("Shutting down.\n"));
```

```
145     fclose(logfile);
146 }
147
148 void writelog(_TCHAR* format, ...) {
149     _TCHAR buf[255];
150     va_list ap;
151
152     struct tm newtime;
153     __time64_t long_time;
154
155     // Get time as 64-bit integer.
156     _time64(&long_time);
157     // Convert to local time.
158     _localtime64_s(&newtime, &long_time);
159
160     // Convert to normal representation.
161     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
162         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
163         newtime.tm_min, newtime.tm_sec);
164
165     // Write date and time
166     fwprintf(logfile, _T("%s"), buf);
167     // Write all params
168     va_start(ap, format);
169     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
170     fwprintf(logfile, _T("%s"), buf);
171     va_end(ap);
172     // New sting
173     fwprintf(logfile, _T("\n"));
174 }
175
176 void syslog(WORD category, WORD identifier, LPWSTR message) {
177     LPWSTR pMessages[1] = { message };
178
179     if (!ReportEvent(
180         eventlog,                   // event log handle
181         EVENTLOG_INFORMATION_TYPE,  // event type
182         category,                   // event category
183         identifier,                 // event identifier
184         NULL,                       // user security identifier
185         1,                          // number of substitution strings
186         0,                          // data size
187         (LPCWSTR*)pMessages,        // pointer to strings
188         NULL)) {                    // pointer to binary data buffer
189         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
190     }
191 }
```

Листинг 6.4: RaiseException.cpp

Результаты выполнения:

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
    ↪ RaiseException\Debug>RaiseException.exe -d
2 ExceptionCode: c0000094, ExceptionFlags: 0, ExceptionRecord: 0,
    ↪ ExceptionAddress: 773ba9f2
3 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
    ↪ RaiseException\Debug>RaiseException.exe -o
4 ExceptionCode: c0000091, ExceptionFlags: 0, ExceptionRecord: 0,
    ↪ ExceptionAddress: 773ba9f2
```

Листинг 6.5: Лог консоли при запуске RaiseException.exe

```
1  [7/10/2017 20:16:39] RaiseException.exe is starting.
2  [7/10/2017 20:16:39] Task: DIVIDE_BY_ZERO exception.
3  [7/10/2017 20:16:39] Ready for generate DIVIDE_BY_ZERO exception.
4  [7/10/2017 20:16:39] ExceptionCode: c0000094, ExceptionFlags: 0,
       ↪ ExceptionRecord: 0, ExceptionAddress: 773ba9f2
5  [7/10/2017 20:16:39] Shutting down.
6
7  [7/10/2017 20:16:41] RaiseException.exe is starting.
8  [7/10/2017 20:16:41] Task: FLT_OVERFLOW exception.
9  [7/10/2017 20:16:41] Ready for generate FLT_OVERFLOW exception.
10 [7/10/2017 20:16:41] ExceptionCode: c0000091, ExceptionFlags: 0,
       ↪ ExceptionRecord: 0, ExceptionAddress: 773ba9f2
11 [7/10/2017 20:16:41] Shutting down.
```

Листинг 6.6: Содержимое файла RaiseException.log

В журнале событий также была информация о вызванных исключений.

Далее 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

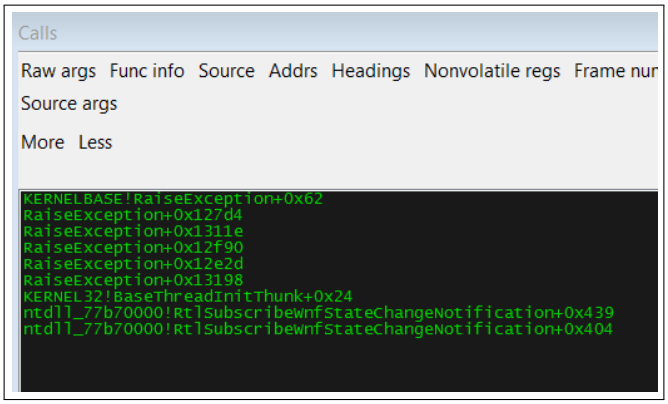



Рис. 6.1: Стек вызовов

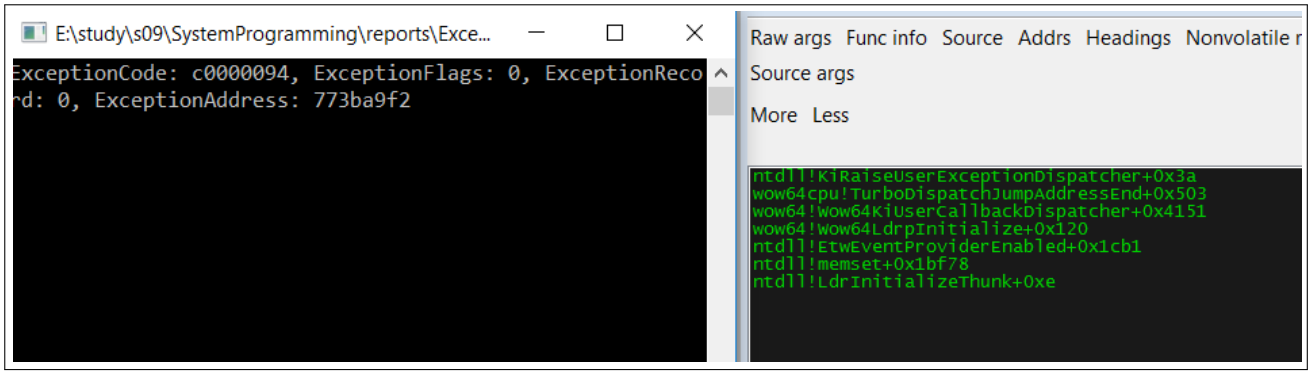



Рис. 6.2: Выведенная информация и очищенный стек

# Необработанные исключения

Если в программе произошло исключение, для которого не существует обработчика исключений, то в этом случае вызывается функция-фильтр системного обработчика исключений, которая выводит на экран окно сообщений с предложением пользователю закончить программу аварийно или выполнить Отладку Приложения. Системная функция-фильтр UnhandledExceptionFilter имеет следующий прототип:

**LONG UnhandledExceptionFilter(PEXCEPTION_POINTERS pExceptionInfo);**

Эта функция имеет один параметр, который указывает на структуру типа EXCEPTION_INFO и возвращает одно из следующих значений:

- EXCEPTION_CONTHSTUE_SEARCH — передать управление отладчику приложения;
- EXCEPTION_EXECUTE_HANDLER — передать управление обработчику исключений.

Приложение может заменить системную функцию-фильтр с помощью функции SetUnhandledExceptionFilter, которая имеет следующий прототип:

**LPTOP_LEVEL_EXCEPTION_FILTER SetUnhandledExceptionFilter(**
**LPTOP_LEVEL_EXCEPTION_FILTER lpTopLevelExceptionFilter);**

Эта функция возвращает адрес старой функции фильтра или null, если установлен системный обработчик исключений. Единственным параметром этой функции является указатель на новую функцию-фильтр, которая будет установлена вместо системной. Эта функция-фильтр должна иметь прототип, соответствующий системной функции фильтра UnhandledExceptionFilter, и возвращать одно из следующих значений:

- EXCEPTION_EXECUTE_HANDLER — выполнение программы прекращается;
- EXCEPTION_CONTINUE_EXECUTION — возобновить исполнение программы с точки исключения;
- EXCEPTION_CONTINUE_SEARCH — выполняется системная функция UnhandledExceptionFilter.

Для того чтобы восстановить системную функцию-фильтр UnhandledExceptionFilter, нужно вызвать функцию UnhandledExceptionFilter с параметром null.

В далее приведённой программе, создана функция-фильтр для обработки всех исключений, для которых не указан другой обработчик.

```
1 #include <stdio.h>
2 #include <tchar.h>
3 #include <cstring>
4 #include <cfloat>
5 #include <cmath>
6 #include <excpt.h>
```

```
 7 #include <windows.h>
 8 #include <time.h>
 9
10 #include "messages.h"
11
12 // log
13 FILE* logfile;
14 HANDLE eventlog;
15
16 void usage(const _TCHAR* prog);
17 void initlog(const _TCHAR* prog);
18 void closelog();
19 void writelog(_TCHAR* format, ...);
20 void syslog(WORD category, WORD identifier, LPWSTR message);
21
22 // protype
23 LONG WINAPI MyUnhandledExceptionFilter(EXCEPTION_POINTERS* ExceptionInfo);
24
25 // Task switcher
26 enum {
27     DIVIDE_BY_ZERO,
28     FLT_OVERFLOW
29 } task;
30
31 // Defines the entry point for the console application.
32 int _tmain(int argc, _TCHAR* argv[]) {
33     //Init log
34     initlog(argv[0]);
35     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
36
37     // Check parameters number
38     if (argc != 2) {
39         _tprintf(_T("Too few parameters.\n\n"));
40         writelog(_T("Too few parameters."));
41         usage(argv[0]);
42         closelog();
43         exit(1);
44     }
45
46     // Set task
47     if (!_tcscmp(_T("-d"), argv[1])) {
48         task = DIVIDE_BY_ZERO;
49         writelog(_T("Task: DIVIDE_BY_ZERO exception."));
50     }
51     else if (!_tcscmp(_T("-o"), argv[1])) {
52         task = FLT_OVERFLOW;
53         writelog(_T("Task: FLT_OVERFLOW exception."));
54     }
55     else {
56         _tprintf(_T("Can't parse parameters.\n\n"));
57         writelog(_T("Can't parse parameters."));
58         usage(argv[0]);
59         closelog();
60         exit(1);
61     }
62
63     // Floating point exceptions are masked by default.
64     _clearfp();
65     _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
66
```

```
67      volatile float tmp = 0;
68      ::SetUnhandledExceptionFilter(MyUnhandledExceptionFilter);
69
70      switch (task) {
71      case DIVIDE_BY_ZERO:
72          // throw an exception using the RaiseException() function
73          writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
74          syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
75              _T("Ready for generate DIVIDE_BY_ZERO exception."));
76          RaiseException(EXCEPTION_INT_DIVIDE_BY_ZERO,
77              EXCEPTION_EXECUTE_FAULT, 0, NULL);
78          writelog(_T("DIVIDE_BY_ZERO exception is generated."));
79          break;
80      case FLT_OVERFLOW:
81          // throw an exception using the RaiseException() function
82          writelog(_T("Ready for generate FLT_OVERFLOW exception."));
83          syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
84              _T("Ready for generate FLT_OVERFLOW exception."));
85          RaiseException(EXCEPTION_FLT_OVERFLOW,
86              EXCEPTION_EXECUTE_FAULT, 0, NULL);
87          writelog(_T("Task: FLT_OVERFLOW exception is generated."));
88          break;
89      default:
90          break;
91      }
92
93      closelog();
94      CloseHandle(eventlog);
95      exit(0);
96  }
97
98  LONG WINAPI MyUnhandledExceptionFilter(EXCEPTION_POINTERS* ExceptionInfo) {
99      enum { size = 400 };
100     _TCHAR buf[size] = { '\0' };
101     const _TCHAR* err = _T("Unhandled exception!\nexeption code : 0x");
102     EXCEPTION_RECORD er = *ExceptionInfo->ExceptionRecord;
103     swprintf_s(buf, _T("%s%x%s%x%s%x%s%x"),
104         _T("ExceptionCode: "), er.ExceptionCode,
105         _T(", ExceptionFlags: "), er.ExceptionFlags,
106         _T(", ExceptionRecord: "), er.ExceptionRecord,
107         _T(", ExceptionAddress: "), er.ExceptionAddress,
108         _T(", NumberParameters: "), er.NumberParameters);
109     _tprintf(_T("%s"), buf);
110     writelog(_T("%s"), buf);
111     syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION, buf);
112
113     //return EXCEPTION_CONTINUE_SEARCH;
114     return EXCEPTION_EXECUTE_HANDLER;
115 }
116
117 // Usage manual
118 void usage(const _TCHAR* prog) {
119     _tprintf(_T("Usage: \n"));
120     _tprintf(_T("\t%s -d\n"), prog);
121     _tprintf(_T("\t\t\t for exception float divide by zero,\n"));
122     _tprintf(_T("\t%s -o\n"), prog);
123     _tprintf(_T("\t\t\t for exception float overflow.\n"));
124 }
125
126 void initlog(const _TCHAR* prog) {
```

```
127     _TCHAR logname[255];
128     wcscpy_s(logname, prog);
129 
130     // replace extension
131     _TCHAR* extension;
132     extension = wcsstr(logname, _T(".exe"));
133     wcsncpy_s(extension, 5, _T(".log"), 4);
134 
135     // Try to open log file for append
136     if (_wfopen_s(&logfile, logname, _T("a+"))) {
137         _wperror(_T("The following error occurred"));
138         _tprintf(_T("Can't open log file %s\n"), logname);
139         exit(1);
140     }
141 
142     writelog(_T("%s is starting."), prog);
143 }
144 
145 void closelog() {
146     writelog(_T("Shutting down.\n"));
147     fclose(logfile);
148 }
149 
150 void writelog(_TCHAR* format, ...) {
151     _TCHAR buf[255];
152     va_list ap;
153 
154     struct tm newtime;
155     __time64_t long_time;
156 
157     // Get time as 64-bit integer.
158     _time64(&long_time);
159     // Convert to local time.
160     _localtime64_s(&newtime, &long_time);
161 
162     // Convert to normal representation.
163     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
164         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
165         newtime.tm_min, newtime.tm_sec);
166 
167     // Write date and time
168     fwprintf(logfile, _T("%s"), buf);
169     // Write all params
170     va_start(ap, format);
171     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
172     fwprintf(logfile, _T("%s"), buf);
173     va_end(ap);
174     // New sting
175     fwprintf(logfile, _T("\n"));
176     fflush(logfile);
177 }
178 
179 void syslog(WORD category, WORD identifier, LPWSTR message) {
180     LPWSTR pMessages[1] = { message };
181 
182     if (!ReportEvent(
183         eventlog,                   // event log handle
184         EVENTLOG_INFORMATION_TYPE,  // event type
185         category,                   // event category
186         identifier,                 // event identifier
```

```
187             NULL,                       // user security identifier
188             1,                          // number of substitution strings
189             0,                          // data size
190             (LPCWSTR*)pMessages,        // pointer to strings
191             NULL)) {                    // pointer to binary data buffer
192             writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
193         }
194     }
```

Листинг 7.1: UnhandleExceptionFilter.cpp

Написанный фильтр позволяет возвращать EXCEPTION_CONTINUE_SEARCH или EXCEPTION_EXECUTE_HANDLER, от чего и соответственно отличаются результаты.

Для начала рассмотрим работу EXCEPTION_EXECUTE_HANDLER.

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
    ↪ UnhandleExceptionFilter\Debug>UnhandleExceptionFilter.exe -d
2 ExceptionCode: c0000094, ExceptionFlags: 0, ExceptionRecord: 0,
    ↪ ExceptionAddress: 773ba9f2
3 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
    ↪ UnhandleExceptionFilter\Debug>UnhandleExceptionFilter.exe -o
4 ExceptionCode: c0000091, ExceptionFlags: 0, ExceptionRecord: 0,
    ↪ ExceptionAddress: 773ba9f2
```

Листинг 7.2: Лог консоли при запуске UnhandleExceptionFilter.exe

```
1 [7/10/2017 21:43:48] UnhandleExceptionFilter.exe is starting.
2 [7/10/2017 21:43:48] Task: DIVIDE_BY_ZERO exception.
3 [7/10/2017 21:43:48] Ready for generate DIVIDE_BY_ZERO exception.
4 [7/10/2017 21:43:48] ExceptionCode: c0000094, ExceptionFlags: 0,
    ↪ ExceptionRecord: 0, ExceptionAddress: 773ba9f2
5
6 [7/10/2017 21:43:50] UnhandleExceptionFilter.exe is starting.
7 [7/10/2017 21:43:50] Task: FLT_OVERFLOW exception.
8 [7/10/2017 21:43:50] Ready for generate FLT_OVERFLOW exception.
9 [7/10/2017 21:43:50] ExceptionCode: c0000091, ExceptionFlags: 0,
    ↪ ExceptionRecord: 0, ExceptionAddress: 773ba9f2
```

Листинг 7.3: Содержимое файла UnhandleExceptionFilter.log

Теперь изменим возвращаемое фильтром значения на EXCEPTION_CONTINUE_SEARCH.

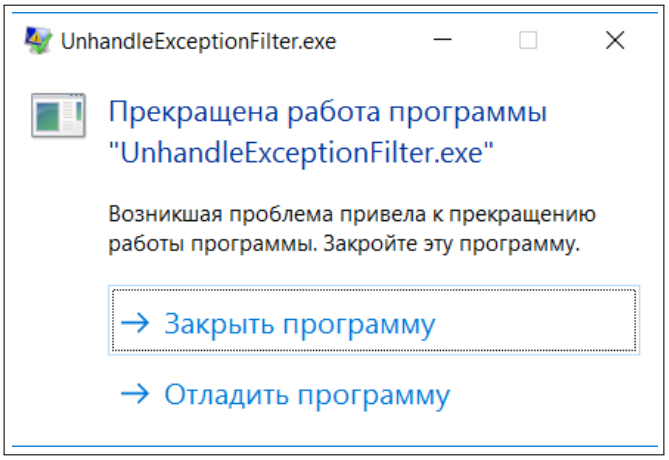



Рис. 7.1: Сообщение о прекращении работы

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
    ↪ UnhandleExceptionFilter\Debug>UnhandleExceptionFilter.exe -d
2 ExceptionCode: c0000094, ExceptionFlags: 0, ExceptionRecord: 0,
    ↪ ExceptionAddress: 773ba9f2
```

Листинг 7.4: Лог консоли при запуске UnhandleExceptionFilter.exe

```
1 [7/10/2017 21:45:22] UnhandleExceptionFilter.exe is starting.
2 [7/10/2017 21:45:22] Task: DIVIDE_BY_ZERO exception.
3 [7/10/2017 21:45:22] Ready for generate DIVIDE_BY_ZERO exception.
4 [7/10/2017 21:45:22] ExceptionCode: c0000094, ExceptionFlags: 0,
    ↪ ExceptionRecord: 0, ExceptionAddress: 773ba9f2
```

Листинг 7.5: Содержимое файла UnhandleExceptionFilter.log

В данной ситуации, исключение корректно обработалось фильтром: запись информации в лог-файл и создания события в журнале событий. Однако работа программы была прекращена.

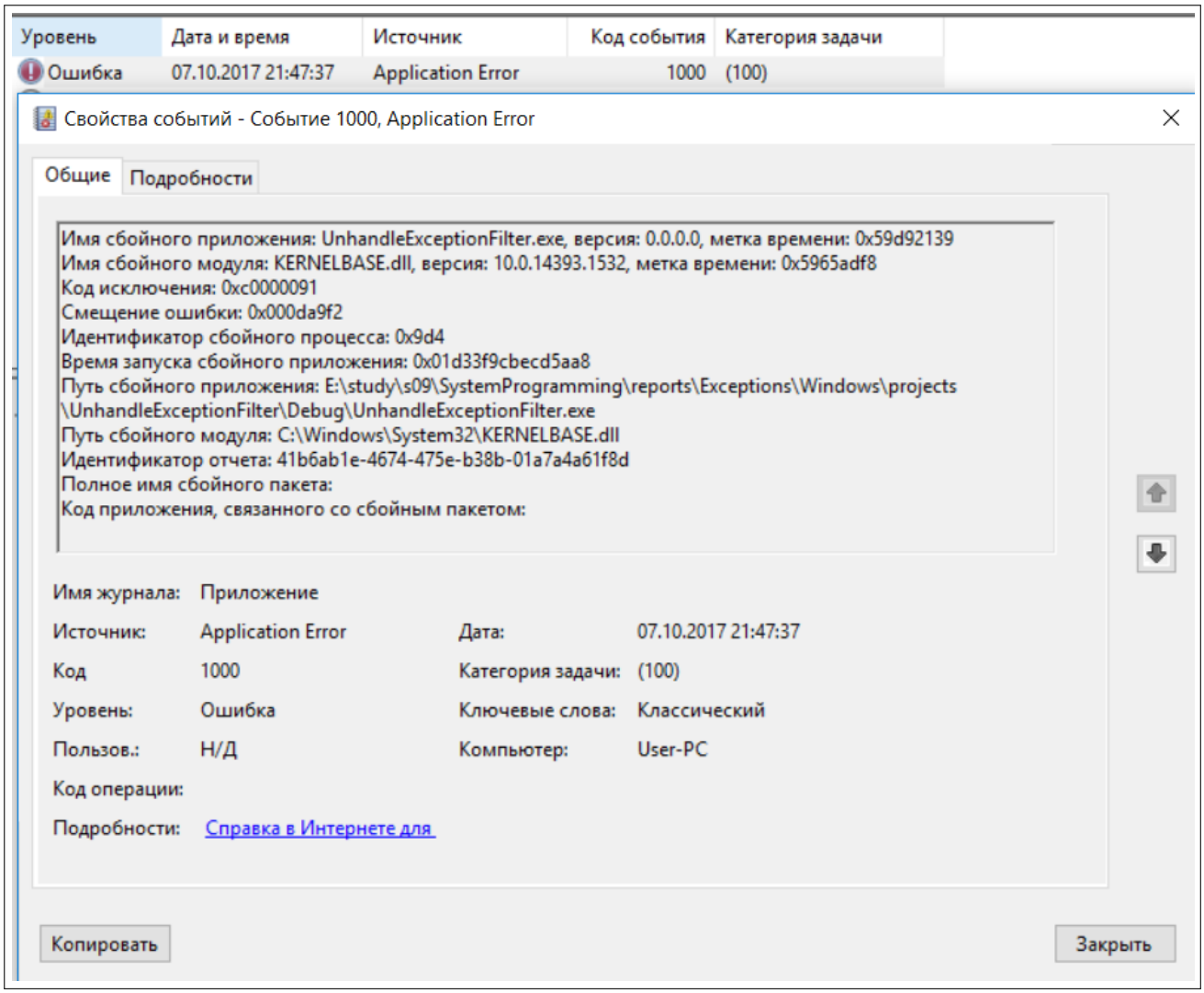



Рис. 7.2: Событие(созданное ОС) о прекращении работы

EXCEPTION_CONTINUE_SEARCH возвращает исключение операционной системе для обработки, что соответственно и вызвало сообщение о прекращении работы и запись в журнале событий.

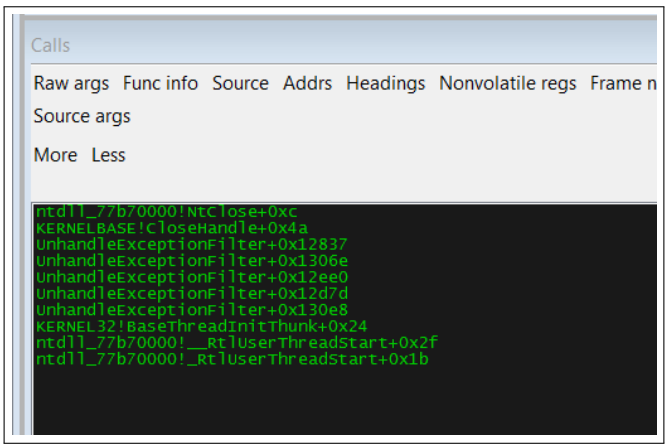



Рис. 7.3: Стек вызовов

Из вышеприведенного эксперимента, можно сделать вывод что возврат EXCEPTION_EXECUTE_HANDLER является более приоритетным в написании собственных программ.

# Вложенные исключения

Далее написана программа, которая показывает как происходит поиск подходящего обработчика исключения. Так в строке 35 листинга 8.1 вызывается исключение деления на ноль, в строке 39 имеется обработчик, но он обрабатывает исключения другого типа, далее имеется обработчик в строке 48, который обрабатывает вызванное исключение.

```
1  #include <stdio.h>
2  #include <tchar.h>
3  #include <cstring>
4  #include <cfloat>
5  #include <excpt.h>
6  #include <windows.h>
7  #include <time.h>
8
9  #include "messages.h"
10
11 // log
12 FILE* logfile;
13 HANDLE eventlog;
14
15 void initlog(const _TCHAR* prog);
16 void closelog();
17 void writelog(_TCHAR* format, ...);
18 void syslog(WORD category, WORD identifier, LPWSTR message);
19
20 // Defines the entry point for the console application.
21 int _tmain(int argc, _TCHAR* argv[]) {
22     //Init log
23     initlog(argv[0]);
24     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
25
26     // Floating point exceptions are masked by default.
27     _clearfp();
28     _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);
29
30     __try {
31         __try {
32             writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
33             syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
34                 _T("Ready for generate DIVIDE_BY_ZERO exception."));
35             RaiseException(EXCEPTION_INT_DIVIDE_BY_ZERO,
36                 EXCEPTION_NONCONTINUABLE, 0, NULL);
37             writelog(_T("DIVIDE_BY_ZERO exception is generated."));
38         }
39         __except ((GetExceptionCode() == EXCEPTION_FLT_OVERFLOW) ?
40             EXCEPTION_EXECUTE_HANDLER :
41             EXCEPTION_CONTINUE_SEARCH){
42             writelog(_T("Internal handler in action."));
43             _tprintf(_T("Internal handler in action."));
```

```
44              syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
45                  _T("Internal handler in action."));
46          }
47      }
48      __except ((GetExceptionCode() == EXCEPTION_FLT_DIVIDE_BY_ZERO) ?
49          EXCEPTION_EXECUTE_HANDLER :
50          EXCEPTION_CONTINUE_SEARCH)
51      {
52          writelog(_T("External handler in action."));
53          _tprintf(_T("External handler in action."));
54          syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
55              _T("Internal handler in action."));
56      }
57
58      closelog();
59      CloseHandle(eventlog);
60      exit(0);
61  }
62
63  void initlog(const _TCHAR* prog) {
64      _TCHAR logname[255];
65      wcscpy_s(logname, prog);
66
67      // replace extension
68      _TCHAR* extension;
69      extension = wcsstr(logname, _T(".exe"));
70      wcsncpy_s(extension, 5, _T(".log"), 4);
71
72      // Try to open log file for append
73      if (_wfopen_s(&logfile, logname, _T("a+"))) {
74          _wperror(_T("The following error occurred"));
75          _tprintf(_T("Can't open log file %s\n"), logname);
76          exit(1);
77      }
78
79      writelog(_T("%s is starting."), prog);
80  }
81
82  void closelog() {
83      writelog(_T("Shutting down.\n"));
84      fclose(logfile);
85  }
86
87  void writelog(_TCHAR* format, ...) {
88      _TCHAR buf[255];
89      va_list ap;
90
91      struct tm newtime;
92      __time64_t long_time;
93
94      // Get time as 64-bit integer.
95      _time64(&long_time);
96      // Convert to local time.
97      _localtime64_s(&newtime, &long_time);
98
99      // Convert to normal representation.
100     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
101         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
102         newtime.tm_min, newtime.tm_sec);
103
```

```
104     // Write date and time
105     fwprintf(logfile, _T("%s"), buf);
106     // Write all params
107     va_start(ap, format);
108     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
109     fwprintf(logfile, _T("%s"), buf);
110     va_end(ap);
111     // New sting
112     fwprintf(logfile, _T("\n"));
113 }
114
115 void syslog(WORD category, WORD identifier, LPWSTR message) {
116     LPWSTR pMessages[1] = { message };
117
118     if (!ReportEvent(
119         eventlog,                       // event log handle
120         EVENTLOG_INFORMATION_TYPE,      // event type
121         category,                       // event category
122         identifier,                     // event identifier
123         NULL,                           // user security identifier
124         1,                              // number of substitution strings
125         0,                              // data size
126         (LPCWSTR*)pMessages,            // pointer to strings
127         NULL)) {                        // pointer to binary data buffer
128         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
129     }
130 }
```

Листинг 8.1: NestedException.cpp

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
    ↪ NestedException\Debug>NestedException.exe
2 External handler in action.
```

Листинг 8.2: Лог консоли при запуске NestedException.exe

```
1 [8/10/2017 12:24:32] NestedException.exe is starting.
2 [8/10/2017 12:24:32] Ready for generate DIVIDE_BY_ZERO exception.
3 [8/10/2017 12:24:32] External handler in action.
4 [8/10/2017 12:24:32] Shutting down.
```

Листинг 8.3: Содержимое файла NestedException.log

Как и ожидалось, исключение успешно обработалось во втором обработчике, рассмотрим этот процесс в WinDbg.

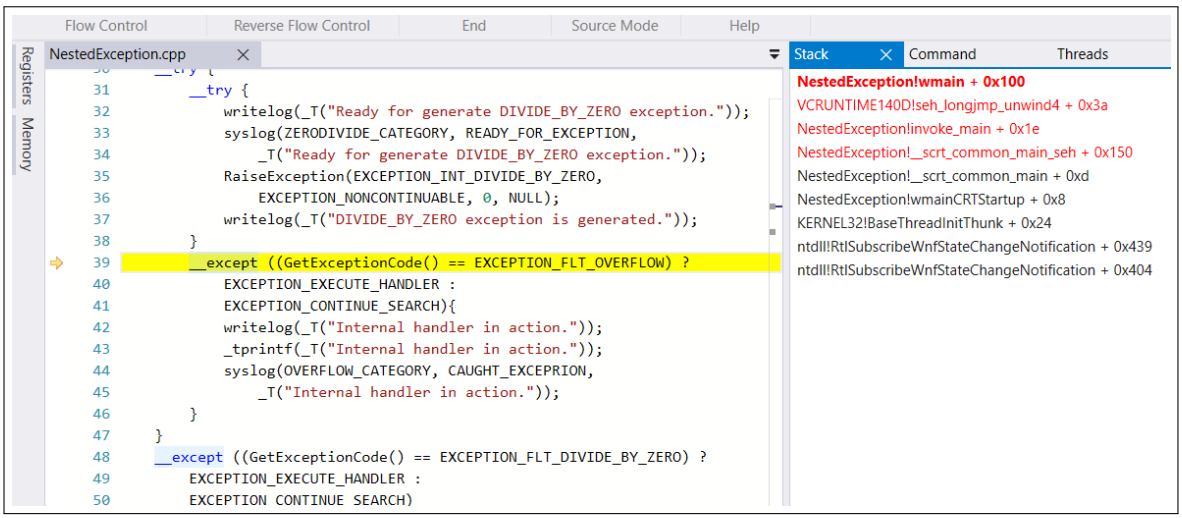



Рис. 8.1: Первый обработчик

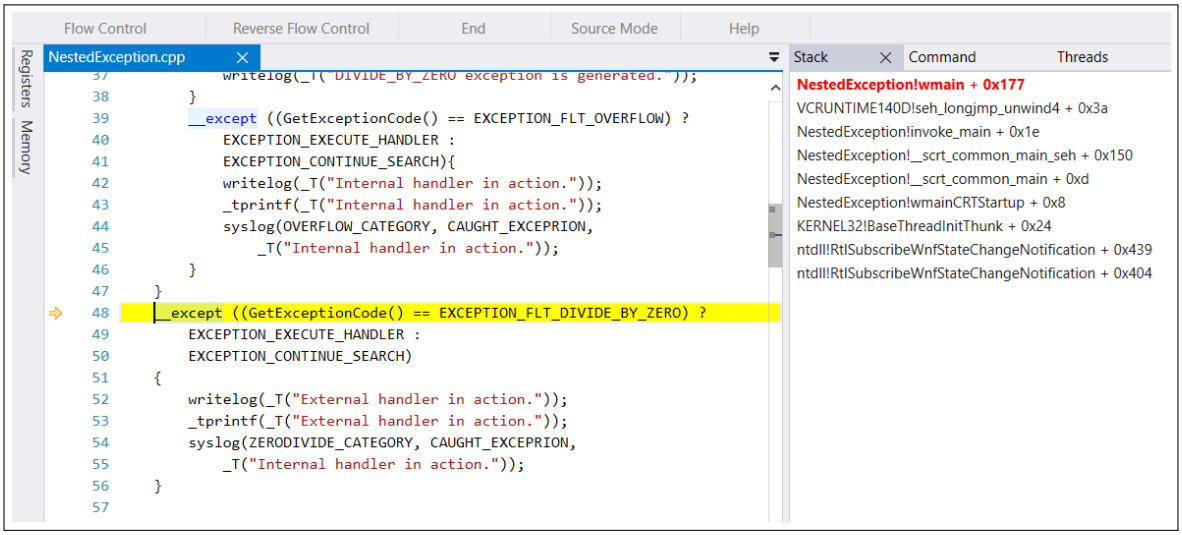



Рис. 8.2: Второй обработчик

Отладчик подтвердил ожидания - поиск подходящего обработчика для исключения происходит снизу вверх. В начале проверяется ближайший обработчик, эта проверка вернёт EXCEPTION_CONTINUE_SEARCH для продолжения поиска более подходящего обработчика и передачи управления дальше по стеку.

# Выход из блока _try с помощью оператора goto

Для передачи управления из фрейма можно использовать оператор goto. При его использовании система считает, что блок с охраняемым кодом завершился аварийно и поэтому выполняет глобальную раскрутку стека. Следовательно, использование инструкции goto вызывает исполнение дополнительного программного кода, что замедляет выполнение программы. Далее приведена программа, которая используют инструкцию goto для выхода из блока _try.

```
1  #include <stdio.h>
2  #include <tchar.h>
3  #include <cstring>
4  #include <cfloat>
5  #include <excpt.h>
6  #include <windows.h>
7  #include <time.h>
8
9  #include "messages.h"
10
11 // log
12 FILE* logfile;
13 HANDLE eventlog;
14
15 void initlog(const _TCHAR* prog);
16 void closelog();
17 void writelog(_TCHAR* format, ...);
18 void syslog(WORD category, WORD identifier, LPWSTR message);
19
20 // Defines the entry point for the console application.
21 int _tmain(int argc, _TCHAR* argv[]) {
22     //Init log
23     initlog(argv[0]);
24     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
25
26     __try {
27         writelog(_T("Call goto"));
28         goto myPoint;
29         writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
30         syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
31             _T("Ready for generate DIVIDE_BY_ZERO exception."));
32         RaiseException(EXCEPTION_INT_DIVIDE_BY_ZERO,
33             EXCEPTION_NONCONTINUABLE, 0, NULL);
34         writelog(_T("DIVIDE_BY_ZERO exception is generated."));
35     }
36     __except (EXCEPTION_EXECUTE_HANDLER)
37     {
38         writelog(_T("Handler in action."));
39         _tprintf(_T("Handler in action."));
```

```
40          syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
41              _T("Handler in action."));
42      }
43  myPoint:
44      writelog(_T("A point outside the __try block."));
45      _tprintf(_T("A point outside the __try block."));
46      syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
47          _T("A point outside the __try block."));
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          _wperror(_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     }
121 }
```

Листинг 9.1: goto.cpp

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\goto\Debug>
    ↪ goto.exe
2 A point outside the __try block.
```

Листинг 9.2: Лог консоли при запуске goto.exe

```
1 [8/10/2017 12:54:35] goto.exe is starting.
2 [8/10/2017 12:54:35] Call goto
3 [8/10/2017 12:54:35] A point outside the __try block.
4 [8/10/2017 12:54:35] Shutting down.
```

Листинг 9.3: Содержимое файла goto.log

Рассмотрим ситуацию в отладчике.

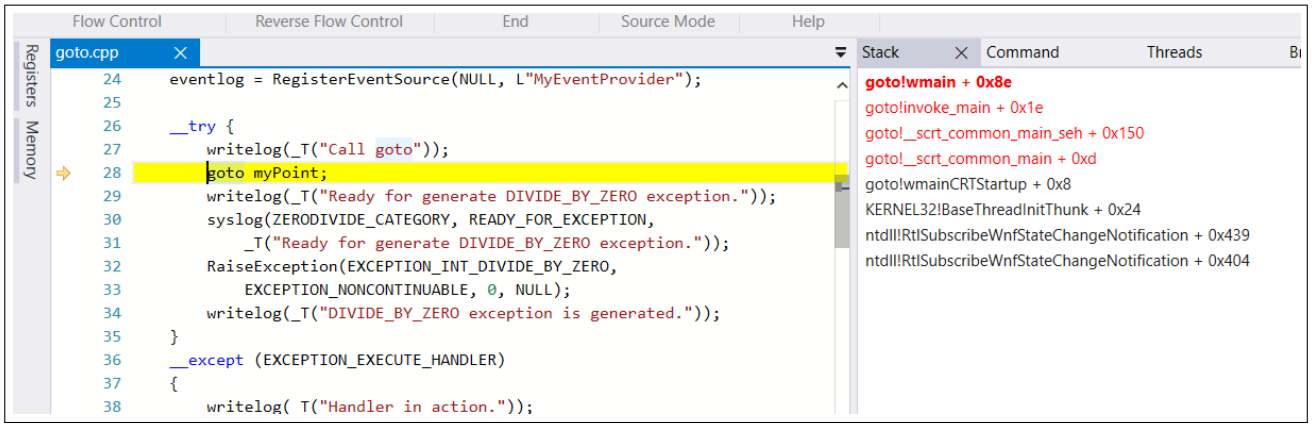



Рис. 9.1: Перед переходом

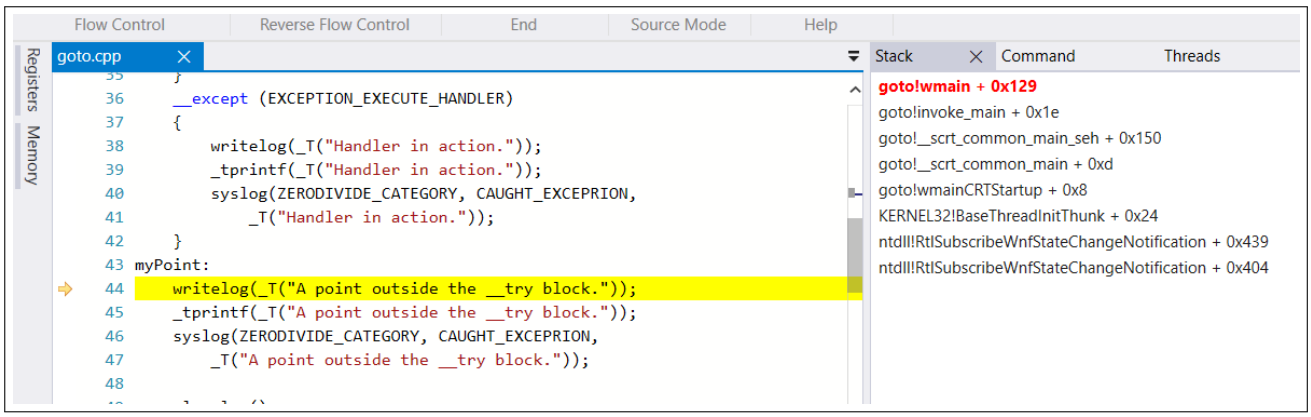



Рис. 9.2: После перехода

Как только была достигнута строка с оператором goto, был выполнен безусловный переход к метке, что позволило не вызывать исключение, логи также подтверждают, что вызова исключения не происходило.

# Выход из блока _try с помощью оператора leave

123

```
#include <stdio.h>
#include <tchar.h>
#include <cstring>
#include <cfloat>
#include <excpt.h>
#include <windows.h>
#include <time.h>

#include "messages.h"

// log
FILE* logfile;
HANDLE eventlog;

void initlog(const _TCHAR* prog);
void closelog();
void writelog(_TCHAR* format, ...);
void syslog(WORD category, WORD identifier, LPWSTR message);

// Defines the entry point for the console application.
int _tmain(int argc, _TCHAR* argv[]) {
    //Init log
    initlog(argv[0]);
    eventlog = RegisterEventSource(NULL, L"MyEventProvider");

    // Floating point exceptions are masked by default.
    _clearfp();
    _controlfp_s(NULL, 0, _EM_OVERFLOW | _EM_ZERODIVIDE);

    __try {
        writelog(_T("Call __leave"));
        __leave;
        writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
        syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
            _T("Ready for generate DIVIDE_BY_ZERO exception."));
        RaiseException(EXCEPTION_INT_DIVIDE_BY_ZERO,
            EXCEPTION_NONCONTINUABLE, 0, NULL);
        writelog(_T("DIVIDE_BY_ZERO exception is generated."));
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        writelog(_T("Handler in action."));
        _tprintf(_T("Handler in action."));
        syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
            _T("Handler in action."));
```

```
46      }
47      writelog(_T("A point outside the __try block."));
48      _tprintf(_T("A point outside the __try block."));
49      syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
50          _T("A point outside the __try block."));
51
52      closelog();
53      CloseHandle(eventlog);
54      exit(0);
55  }
56
57  void initlog(const _TCHAR* prog) {
58      _TCHAR logname[255];
59      wcscpy_s(logname, prog);
60
61      // replace extension
62      _TCHAR* extension;
63      extension = wcsstr(logname, _T(".exe"));
64      wcsncpy_s(extension, 5, _T(".log"), 4);
65
66      // Try to open log file for append
67      if (_wfopen_s(&logfile, logname, _T("a+"))) {
68          _wperror(_T("The following error occurred"));
69          _tprintf(_T("Can't open log file %s\n"), logname);
70          exit(1);
71      }
72
73      writelog(_T("%s is starting."), prog);
74  }
75
76  void closelog() {
77      writelog(_T("Shutting down.\n"));
78      fclose(logfile);
79  }
80
81  void writelog(_TCHAR* format, ...) {
82      _TCHAR buf[255];
83      va_list ap;
84
85      struct tm newtime;
86      __time64_t long_time;
87
88      // Get time as 64-bit integer.
89      _time64(&long_time);
90      // Convert to local time.
91      _localtime64_s(&newtime, &long_time);
92
93      // Convert to normal representation.
94      swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
95          newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
96          newtime.tm_min, newtime.tm_sec);
97
98      // Write date and time
99      fwprintf(logfile, _T("%s"), buf);
100     // Write all params
101     va_start(ap, format);
102     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
103     fwprintf(logfile, _T("%s"), buf);
104     va_end(ap);
105     // New sting
```

```
106     fwprintf(logfile, _T("\n"));
107 }
108
109 void syslog(WORD category, WORD identifier, LPWSTR message) {
110     LPWSTR pMessages[1] = { message };
111
112     if (!ReportEvent(
113         eventlog,                   // event log handle
114         EVENTLOG_INFORMATION_TYPE,  // event type
115         category,                   // event category
116         identifier,                 // event identifier
117         NULL,                       // user security identifier
118         1,                          // number of substitution strings
119         0,                          // data size
120         (LPCWSTR*)pMessages,        // pointer to strings
121         NULL)) {                    // pointer to binary data buffer
122         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
123     }
124 }
```

Листинг 10.1: leave.cpp

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\leave\Debug>
    ↪ leave.exe
2 A point outside the __try block.
```

Листинг 10.2: Лог консоли при запуске leave.exe

```
1 [8/10/2017 13:17:30] leave.exe is starting.
2 [8/10/2017 13:17:30] Call __leave
3 [8/10/2017 13:17:30] A point outside the __try block.
4 [8/10/2017 13:17:30] Shutting down.
```

Листинг 10.3: Содержимое файла leave.log

Рассмотрим ситуацию в отладчике.

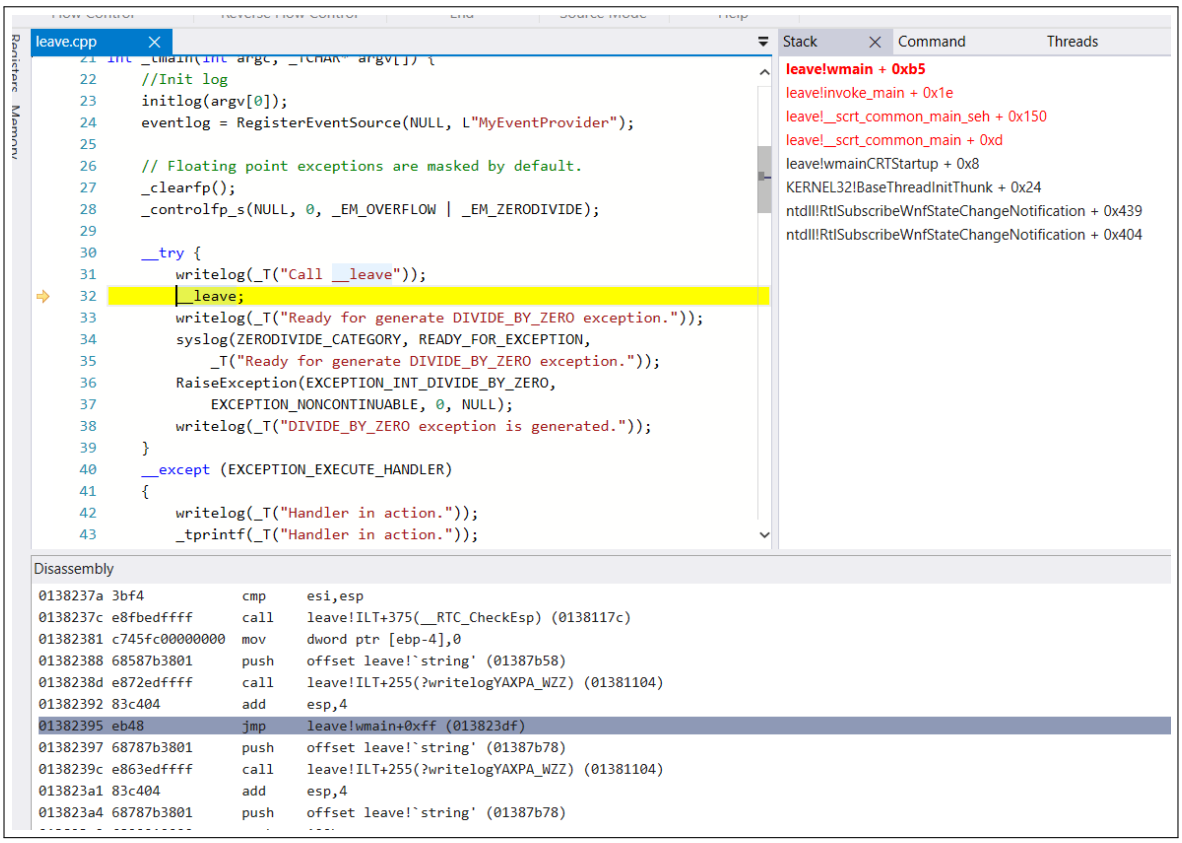



Рис. 10.1: До вызова команды leave

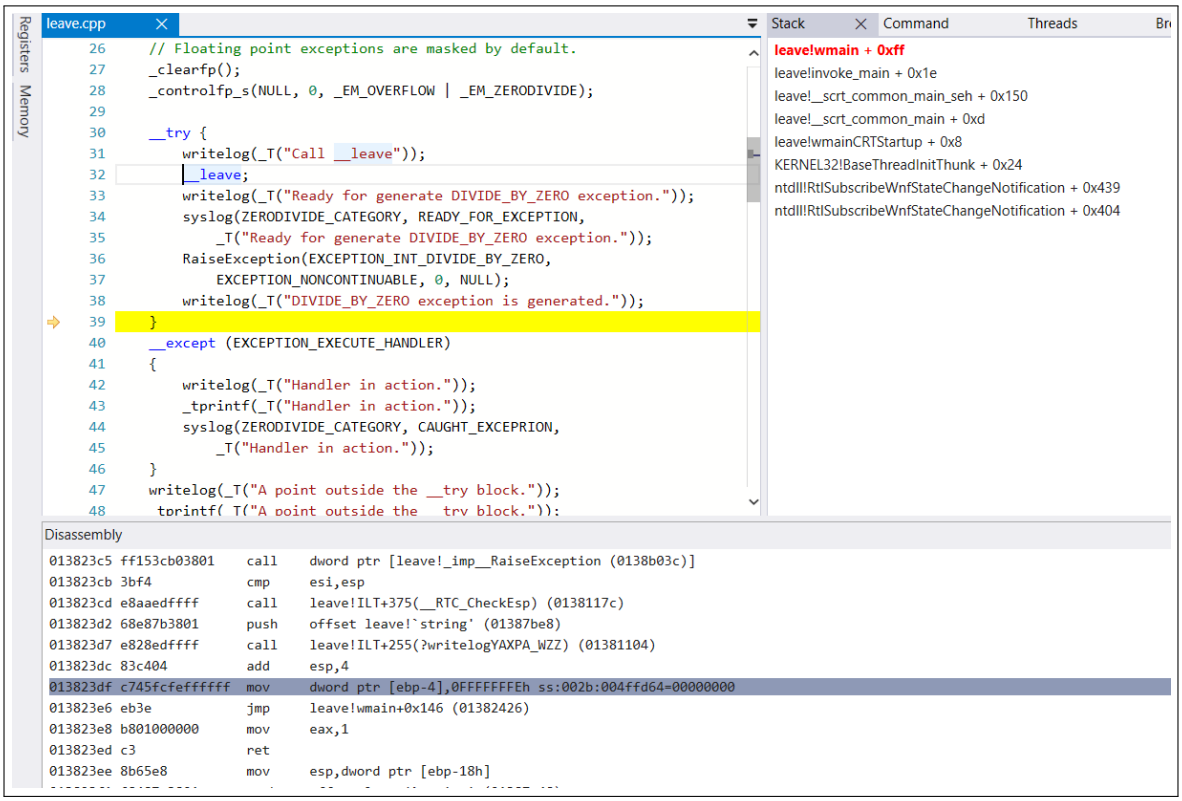



Рис. 10.2: Конец блока _try

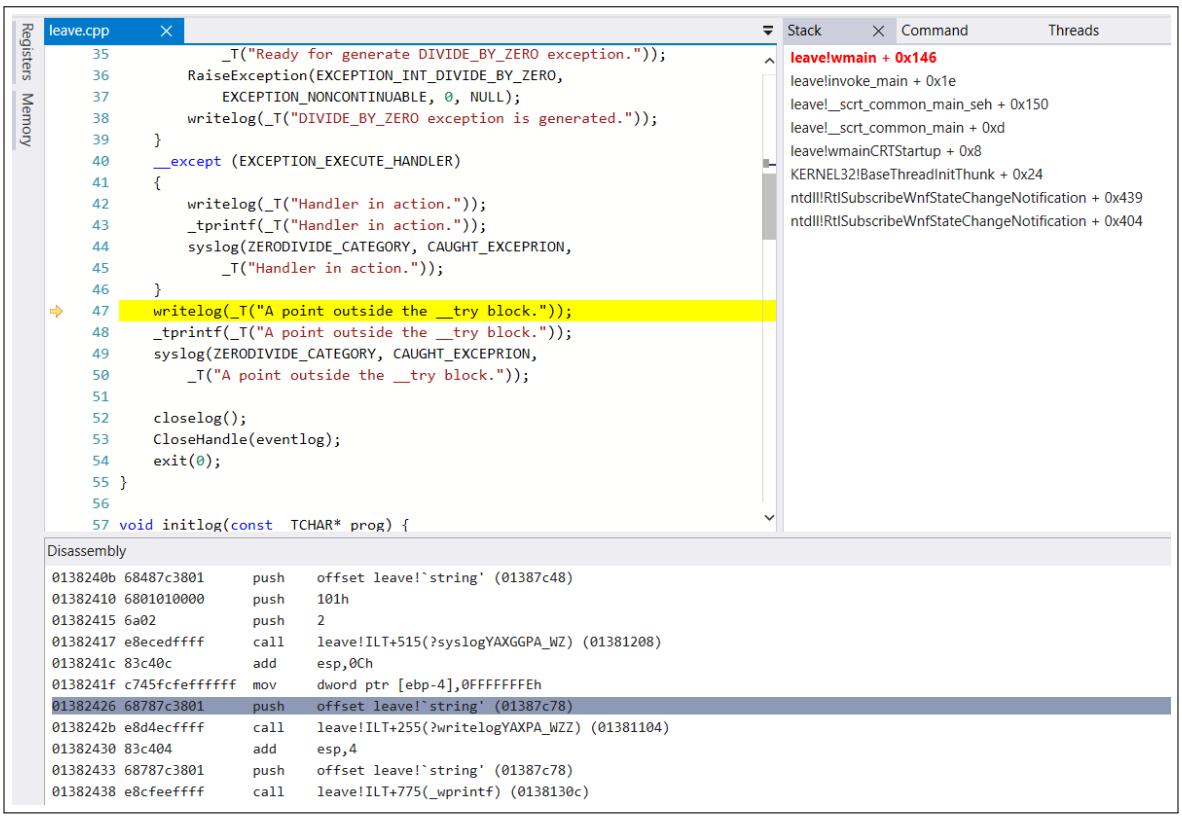



Рис. 10.3: Выход конструкции _try _except

Результат использования _leave — переход в конец блока try, минуя вызов исключения и блок _except. Как итог, результат прежний (если смотреть на логи), но метод его достижения отличается – этот способ считается более правильным, т.к. не приводит к раскрутке стека.

# Преобразование SEH в C++ исключение

Для преобразования структурное исключения в исключение языка C, имеется функция **_set_se_transiator**. Эта функция устанавливает в системе функцию, которая называется функцией-транслятором. Если функция-транслятор установлена, то она вызывается всегда при выбросе структурного исключения. В функции-трансляторе можно использовать инструкцию throw языка программирования C++, которая будет выбрасывать исключение C++ нужного типа.

Функция-транслятор должна иметь следующий прототип:

**typedef void (*_se_translator_function) (unsigned int, Struct _EXCEPTION_POINTERS*);**

который описан в заголовочном файле eh.h. Как видно из этого описания - функция-транслятор не возвращает значения и получает два параметра: код исключения и указатель на структуру типа _EXCEPTION_POINTERS.

Функция _set_se_transiator, которая используется для установки функции-транслятора, также описана в заголовочном файле eh.h и имеет следующий прототип:

**_se_translator_function _set_se_translator(_se_translator_function se_trans_func);**

Единственным параметром этой функции является указатель на новую функцию-транслятор, а возвращает функция _set_se_transiator адрес старой функции-транслятора, которая в дальнейшем может быть восстановлена при помощи вызова _set_se_transiator.

```
1  #include <stdio.h>
2  #include <tchar.h>
3  #include <cstring>
4  #include <cfloat>
5  #include <stdexcept>
6  #include <excpt.h>
7  #include <windows.h>
8  #include <time.h>
9
10 #include "messages.h"
11
12 // log
13 FILE* logfile;
14 HANDLE eventlog;
15
16 void initlog(const _TCHAR* prog);
17 void closelog();
18 void writelog(_TCHAR* format, ...);
19 void syslog(WORD category, WORD identifier, LPWSTR message);
20
21 void se_trans_func(unsigned code, EXCEPTION_POINTERS *p) {
22     throw code;
23 }
24
25 void fltOverflowException() {
```

```
26      RaiseException(
27          3221225617,           // код исключения переполнения float
28          0,                    // continuable exception
29          0, NULL);             // no arguments
30  }
31
32  void devideByZeroException() {
33      RaiseException( // вызываем исключение
34          EXCEPTION_INT_DIVIDE_BY_ZERO,           // код исключения деления на
        ↪ ноль
35          0,                    // continuable exception
36          0, NULL);             // no arguments
37  }
38
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      writelog(_T("Ready for translator ativation."));
47      _set_se_translator(se_trans_func);
48
49      try {
50          _set_se_translator(se_trans_func);
51          writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
52          syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
53              _T("Ready for generate DIVIDE_BY_ZERO exception."));
54          devideByZeroException();
55          writelog(_T("DIVIDE_BY_ZERO exception is generated."));
56      }
57      catch (unsigned code) {
58          _tprintf(_T("CPP exception(DIVIDE_BY_ZERO): %x\n"), code);
59          writelog(_T("CPP exception(DIVIDE_BY_ZERO): %x"), code);
60          syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION, _T("CPP exception(
        ↪ DIVIDE_BY_ZERO)"));
61      }
62
63      try {
64          writelog(_T("Ready for generate FLT_OVERFLOW exception."));
65          syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
66              _T("Ready for generate FLT_OVERFLOW exception."));
67          fltOverflowException();
68          writelog(_T("FLT_OVERFLOW exception is generated."));
69      }
70      catch (unsigned code) {
71          _tprintf(_T("CPP exception(FLT_OVERFLOW): %x\n"), code);
72          writelog(_T("CPP exception:(FLT_OVERFLOW) %x"), code);
73          syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION, _T("CPP exception(
        ↪ FLT_OVERFLOW)"));
74      }
75
76      closelog();
77      CloseHandle(eventlog);
78      exit(0);
79  }
80
81
82  void initlog(const _TCHAR* prog) {
```

```
83      _TCHAR logname[255];
84      wcscpy_s(logname, prog);
85
86      // replace extension
87      _TCHAR* extension;
88      extension = wcsstr(logname, _T(".exe"));
89      wcsncpy_s(extension, 5, _T(".log"), 4);
90
91      // Try to open log file for append
92      if (_wfopen_s(&logfile, logname, _T("a+"))) {
93          _wperror(_T("The following error occurred"));
94          _tprintf(_T("Can't open log file %s\n"), logname);
95          exit(1);
96      }
97
98      writelog(_T("%s is starting."), prog);
99  }
100
101 void closelog() {
102     writelog(_T("Shutting down.\n"));
103     fclose(logfile);
104 }
105
106 void writelog(_TCHAR* format, ...) {
107     _TCHAR buf[255];
108     va_list ap;
109
110     struct tm newtime;
111     __time64_t long_time;
112
113     // Get time as 64-bit integer.
114     _time64(&long_time);
115     // Convert to local time.
116     _localtime64_s(&newtime, &long_time);
117
118     // Convert to normal representation.
119     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
120         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
121         newtime.tm_min, newtime.tm_sec);
122
123     // Write date and time
124     fwprintf(logfile, _T("%s"), buf);
125     // Write all params
126     va_start(ap, format);
127     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
128     fwprintf(logfile, _T("%s"), buf);
129     va_end(ap);
130     // New sting
131     fwprintf(logfile, _T("\n"));
132 }
133
134 void syslog(WORD category, WORD identifier, LPWSTR message) {
135     LPWSTR pMessages[1] = { message };
136
137     if (!ReportEvent(
138         eventlog,                   // event log handle
139         EVENTLOG_INFORMATION_TYPE,  // event type
140         category,                   // event category
141         identifier,                 // event identifier
142         NULL,                       // user security identifier
```

```
143             1,                              // number of substitution strings
144             0,                              // data size
145             (LPCWSTR*)pMessages,            // pointer to strings
146             NULL)) {                        // pointer to binary data buffer
147             writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
148         }
149     }
```

Листинг 11.1: NestedException.cpp

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\translator\
    ↪ Debug>translator.exe
2 CPP exception(DIVIDE_BY_ZERO): c0000094
3 CPP exception(FLT_OVERFLOW): c0000091
```

Листинг 11.2: Лог консоли при запуске translator.exe

```
1 [8/10/2017 14:31:31] translator.exe is starting.
2 [8/10/2017 14:31:31] Ready for translator ativation.
3 [8/10/2017 14:31:31] Ready for generate DIVIDE_BY_ZERO exception.
4 [8/10/2017 14:31:31] CPP exception(DIVIDE_BY_ZERO): c0000094
5 [8/10/2017 14:31:31] Ready for generate FLT_OVERFLOW exception.
6 [8/10/2017 14:31:31] CPP exception:(FLT_OVERFLOW) c0000091
7 [8/10/2017 14:31:31] Shutting down.
```

Листинг 11.3: Содержимое файла translator.log

Транслятор был успешно задан, и с помощью функции throw успешно вызвано исключение языка C++.

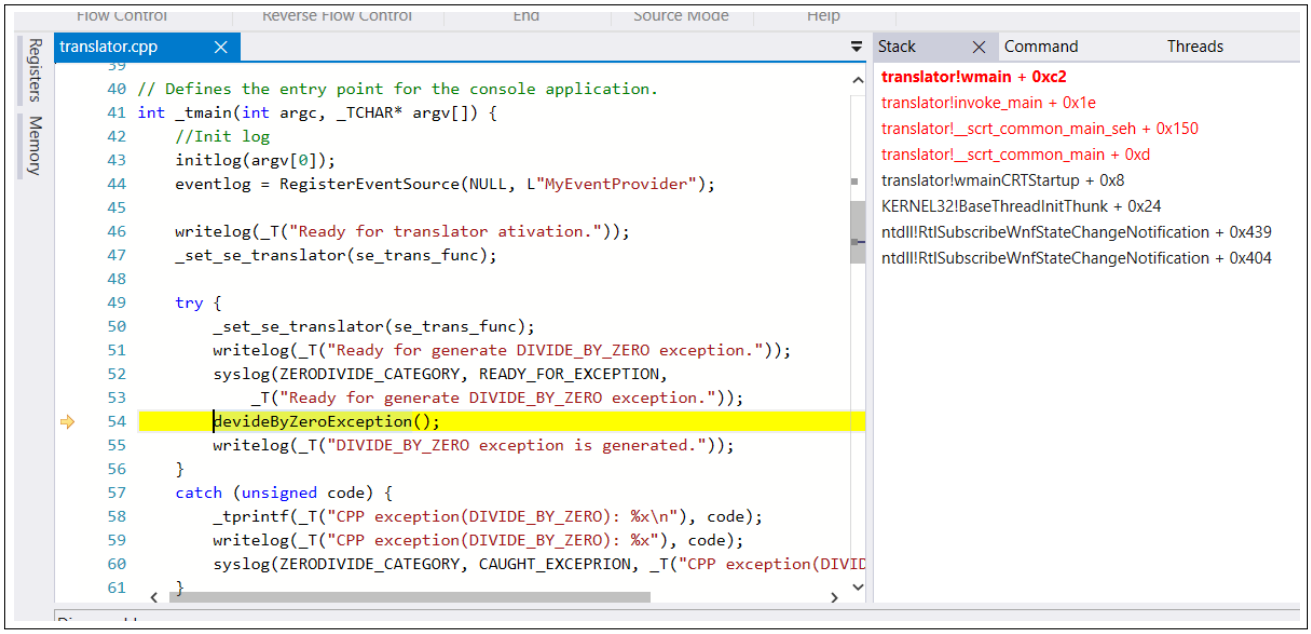



Рис. 11.1: Перед вызовом исключения

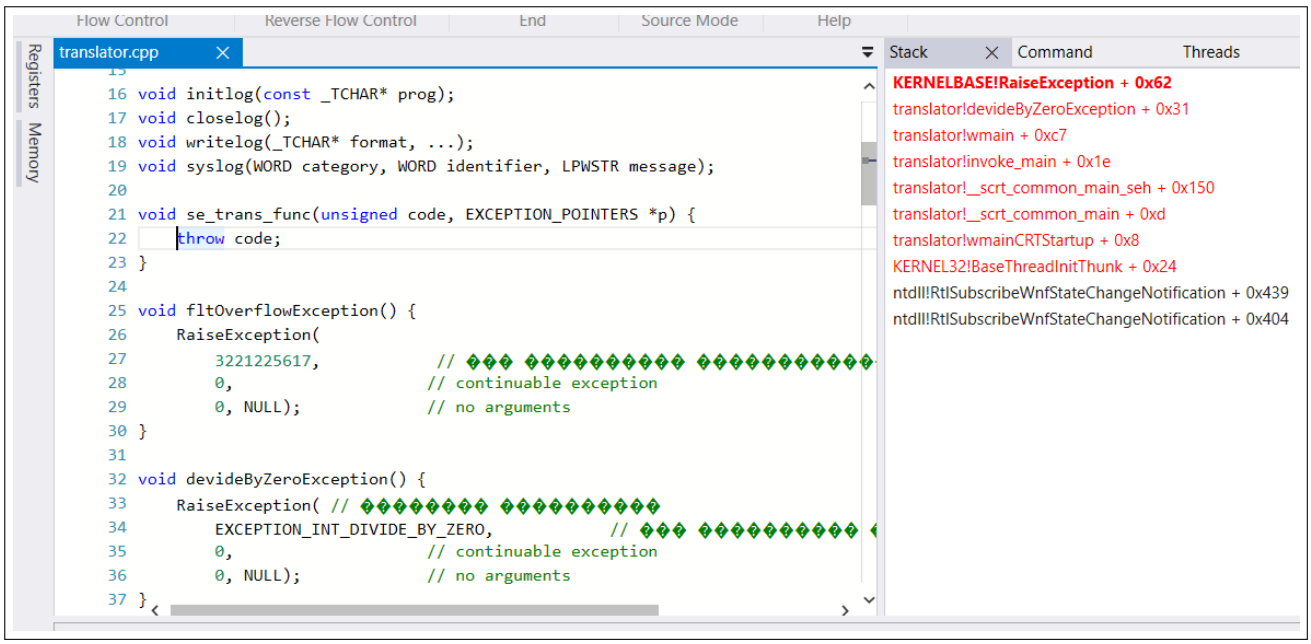



Рис. 11.2: Трансляция в C++ исключение

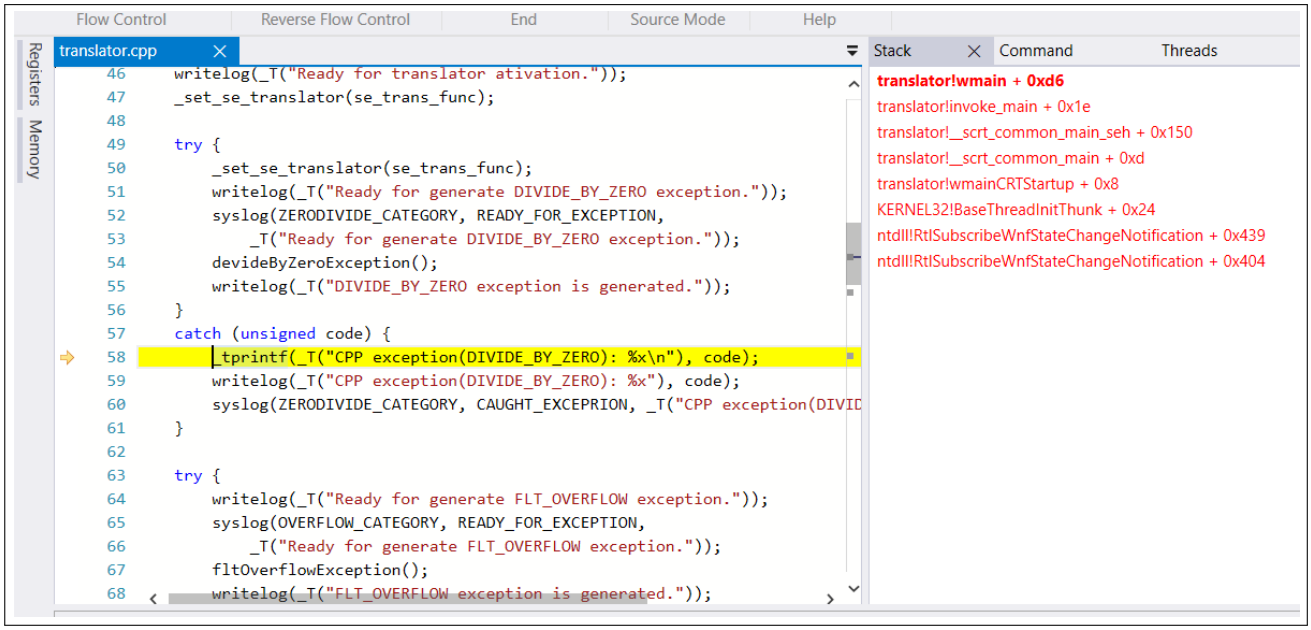



Рис. 11.3: Обработка C++ исключения

Если проследить за передачей управления по стеку вызовов, то сразу после возбуждения исключения в 54-ой строке, управление передаётся транслятору, где генерируется C++ исключения, и только после этого в блок catch, где происходит обработка исключения.

# Финальный обработчик finally

Код, при исполнении которого возможен выброс исключения, как и в случае с фреймовой обработкой исключений, заключается в блок try. Но только теперь за блоком try следует код, который заключается в блок **finally**. Система гарантирует, что при любой передаче управления из блока try, независимо от того, произошло или нет исключение внутри этого блока, предварительно управление будет передано блоку finally. Такой способ обработки исключений называется финальная обработка исключений. Структурно финальная обработка исключений выглядит следующим образом:

**_try**
**{**
**// охраняемый код**
**}**
**finally**
**{**
**// финальный код**
**}**

Финальная обработка исключений используется для того, чтобы при любом исходе исполнения блока _try освободить ресурсы, которые были захвачены внутри этого блока. Такими ресурсами могут быть память, файлы, критические секции и т. д.

Далее приведенная программа, для каждого из исключений имеет 2 блока _try, первый _try _finally, второй _try _except.

```
1  #include <stdio.h>
2  #include <tchar.h>
3  #include <cfloat>
4  #include <excpt.h>
5  #include <time.h>
6  #include <windows.h>
7
8  #include "messages.h"
9
10 // log
11 FILE* logfile;
12 HANDLE eventlog;
13
14 void initlog(const _TCHAR* prog);
15 void closelog();
16 void writelog(_TCHAR* format, ...);
17 void syslog(WORD category, WORD identifier, LPWSTR message);
18
19 void fltOverflowException() {
20     RaiseException(
21         3221225617,           // код исключения переполнения float
22         0,                   // continuable exception
23         0, NULL);            // no arguments
```

```
24 }
25
26 void devideByZeroException() {
27     RaiseException( // вызываем исключение
28         3221225620,           // код исключения деления на ноль
29         0,                    // continuable exception
30         0, NULL);             // no arguments
31 }
32
33
34 // Defines the entry point for the console application.
35 int _tmain(int argc, _TCHAR* argv[]) {
36     //Init log
37     initlog(argv[0]);
38     eventlog = RegisterEventSource(NULL, L"MyEventProvider");
39
40     __try {
41         __try {
42             writelog(_T("Ready for generate EXCEPTION_INT_DIVIDE_BY_ZERO
    ↪ exception."));
43             syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
44                 _T("Ready for generate EXCEPTION_INT_DIVIDE_BY_ZERO exception."
    ↪ ));
45             devideByZeroException();
46             writelog(_T("EXCEPTION_INT_DIVIDE_BY_ZERO exception is generated.")
    ↪ );
47         }
48         __finally
49         {
50             writelog(_T("Finally block for EXCEPTION_INT_DIVIDE_BY_ZERO"));
51             _tprintf(_T("Finally block for EXCEPTION_INT_DIVIDE_BY_ZERO\n"));
52             syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
53                 _T("Finally block for EXCEPTION_INT_DIVIDE_BY_ZERO"));
54         }
55     }
56     __except (GetExceptionCode() == EXCEPTION_INT_DIVIDE_BY_ZERO) {
57         writelog(_T(" Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO"));
58         _tprintf(_T(" Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO\n"));
59         syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
60             _T(" Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO ."));
61     }
62
63     __try {
64         __try {
65             writelog(_T("Ready for generate EXCEPTION_FLT_OVERFLOW exception.")
    ↪ );
66             syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
67                 _T("Ready for generate EXCEPTION_FLT_OVERFLOW exception."));
68             fltOverflowException();
69             writelog(_T("EXCEPTION_FLT_OVERFLOW exception is generated."));
70         }
71         __finally
72         {
73             writelog(_T("Finally block for EXCEPTION_FLT_OVERFLOW"));
74             _tprintf(_T("Finally block for EXCEPTION_FLT_OVERFLOW\n"));
75             syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
76                 _T("Finally block for EXCEPTION_FLT_OVERFLOW"));
77         }
78     }
79     __except (GetExceptionCode() == EXCEPTION_FLT_OVERFLOW) {
```

```
80          writelog(_T(" Caught exception is: EXCEPTION_FLT_OVERFLOW"));
81          _tprintf(_T(" Caught exception is: EXCEPTION_FLT_OVERFLOW\n"));
82          syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
83              _T(" Caught exception is: EXCEPTION_FLT_OVERFLOW ."));
84      }
85
86
87      closelog();
88      CloseHandle(eventlog);
89      exit(0);
90  }
91
92  void initlog(const _TCHAR* prog) {
93      _TCHAR logname[255];
94      wcscpy_s(logname, prog);
95
96      // replace extension
97      _TCHAR* extension;
98      extension = wcsstr(logname, _T(".exe"));
99      wcsncpy_s(extension, 5, _T(".log"), 4);
100
101     // Try to open log file for append
102     if (_wfopen_s(&logfile, logname, _T("a+"))) {
103         _wperror(_T("The following error occurred"));
104         _tprintf(_T("Can't open log file %s\n"), logname);
105         exit(1);
106     }
107
108     writelog(_T("%s is starting."), prog);
109 }
110
111 void closelog() {
112     writelog(_T("Shutting down.\n"));
113     fclose(logfile);
114 }
115
116 void writelog(_TCHAR* format, ...) {
117     _TCHAR buf[255];
118     va_list ap;
119
120     struct tm newtime;
121     __time64_t long_time;
122
123     // Get time as 64-bit integer.
124     _time64(&long_time);
125     // Convert to local time.
126     _localtime64_s(&newtime, &long_time);
127
128     // Convert to normal representation.
129     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
130         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
131         newtime.tm_min, newtime.tm_sec);
132
133     // Write date and time
134     fwprintf(logfile, _T("%s"), buf);
135     // Write all params
136     va_start(ap, format);
137     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
138     fwprintf(logfile, _T("%s"), buf);
139     va_end(ap);
```

```
140     // New sting
141     fwprintf(logfile, _T("\n"));
142 }
143
144 void syslog(WORD category, WORD identifier, LPWSTR message) {
145     LPWSTR pMessages[1] = { message };
146
147     if (!ReportEvent(
148         eventlog,                       // event log handle
149         EVENTLOG_INFORMATION_TYPE,      // event type
150         category,                       // event category
151         identifier,                     // event identifier
152         NULL,                           // user security identifier
153         1,                              // number of substitution strings
154         0,                              // data size
155         (LPCWSTR*)pMessages,            // pointer to strings
156         NULL)) {                        // pointer to binary data buffer
157         writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
158     }
159 }
```

Листинг 12.1: finally.cpp

```
1 E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\finally\
    ↪ Debug>finally.exe
2 Finally block for EXCEPTION_INT_DIVIDE_BY_ZERO
3 Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO
4 Finally block for EXCEPTION_FLT_OVERFLOW
5 Caught exception is: EXCEPTION_FLT_OVERFLOW
```

Листинг 12.2: Лог консоли при запуске finally.exe

```
1 [8/10/2017 15:16:20] finally.exe is starting.
2 [8/10/2017 15:16:20] Ready for generate EXCEPTION_INT_DIVIDE_BY_ZERO exception.
3 [8/10/2017 15:16:20] Finally block for EXCEPTION_INT_DIVIDE_BY_ZERO
4 [8/10/2017 15:16:20] Caught exception is: EXCEPTION_INT_DIVIDE_BY_ZERO
5 [8/10/2017 15:16:20] Ready for generate EXCEPTION_FLT_OVERFLOW exception.
6 [8/10/2017 15:16:20] Finally block for EXCEPTION_FLT_OVERFLOW
7 [8/10/2017 15:16:20] Caught exception is: EXCEPTION_FLT_OVERFLOW
8 [8/10/2017 15:16:20] Shutting down.
```

Листинг 12.3: Содержимое файла finally.log

Как можно заметить из логов, сперва выполняется блок _finally, а затем _except.

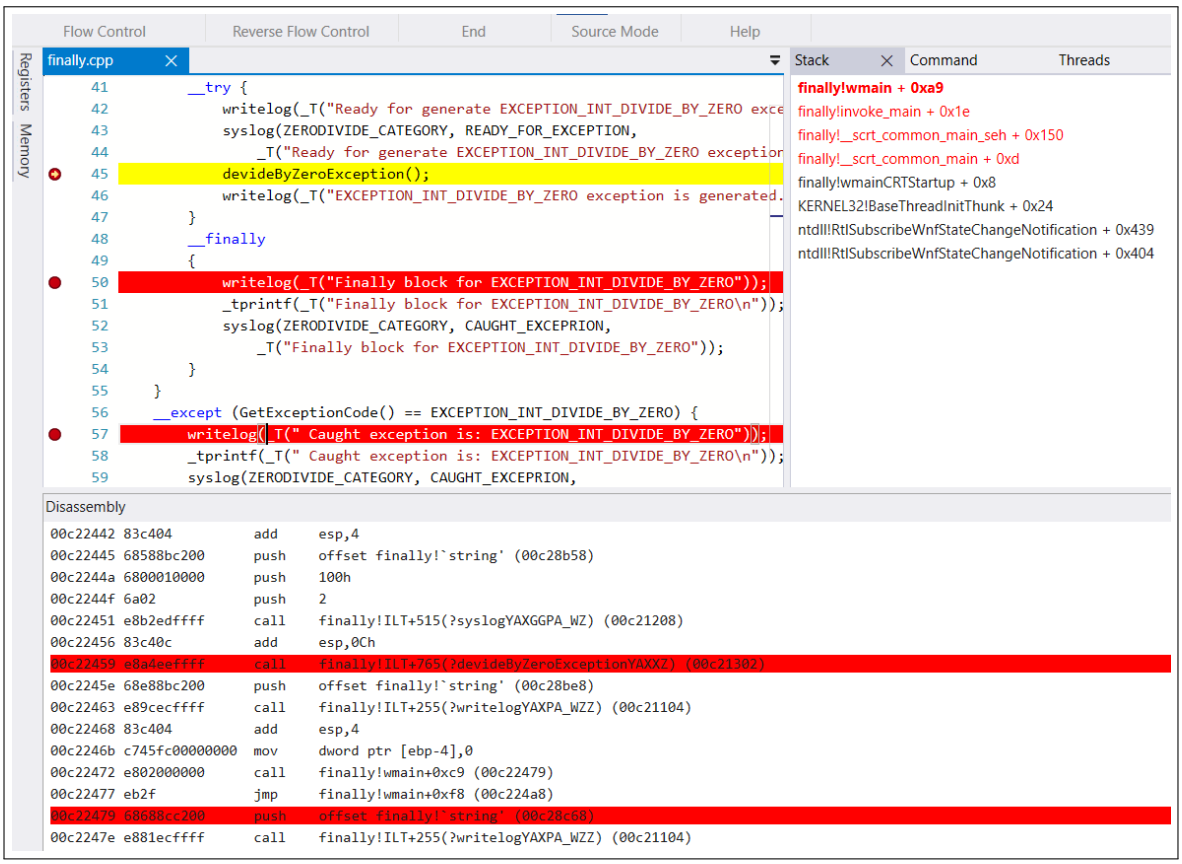



Рис. 12.1: Перед вызовом исключения

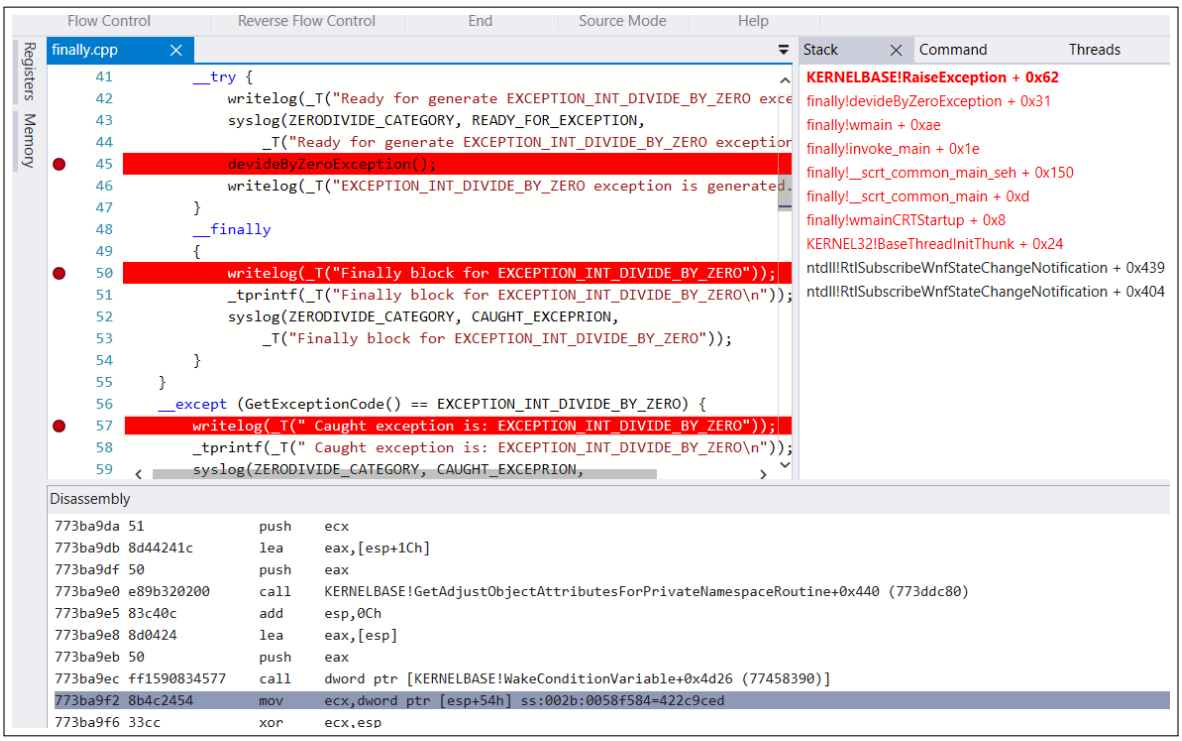



Рис. 12.2: Вызов исключения

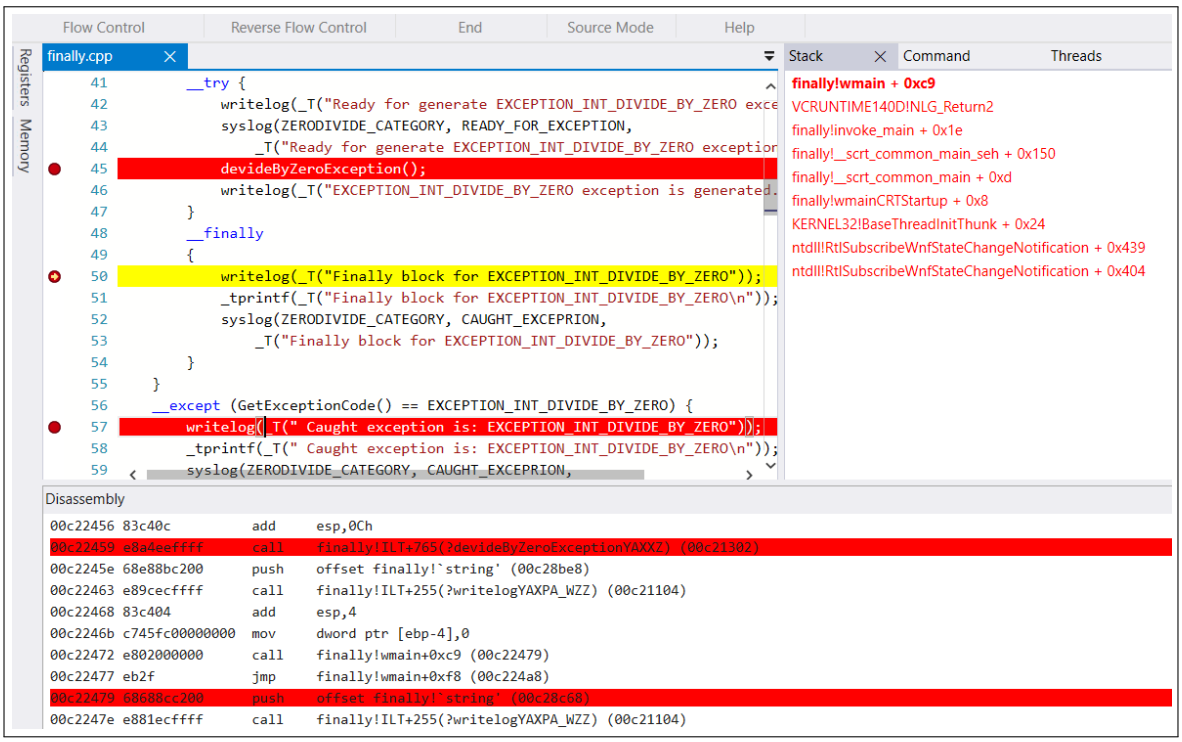



Рис. 12.3: Вызов блока _finally

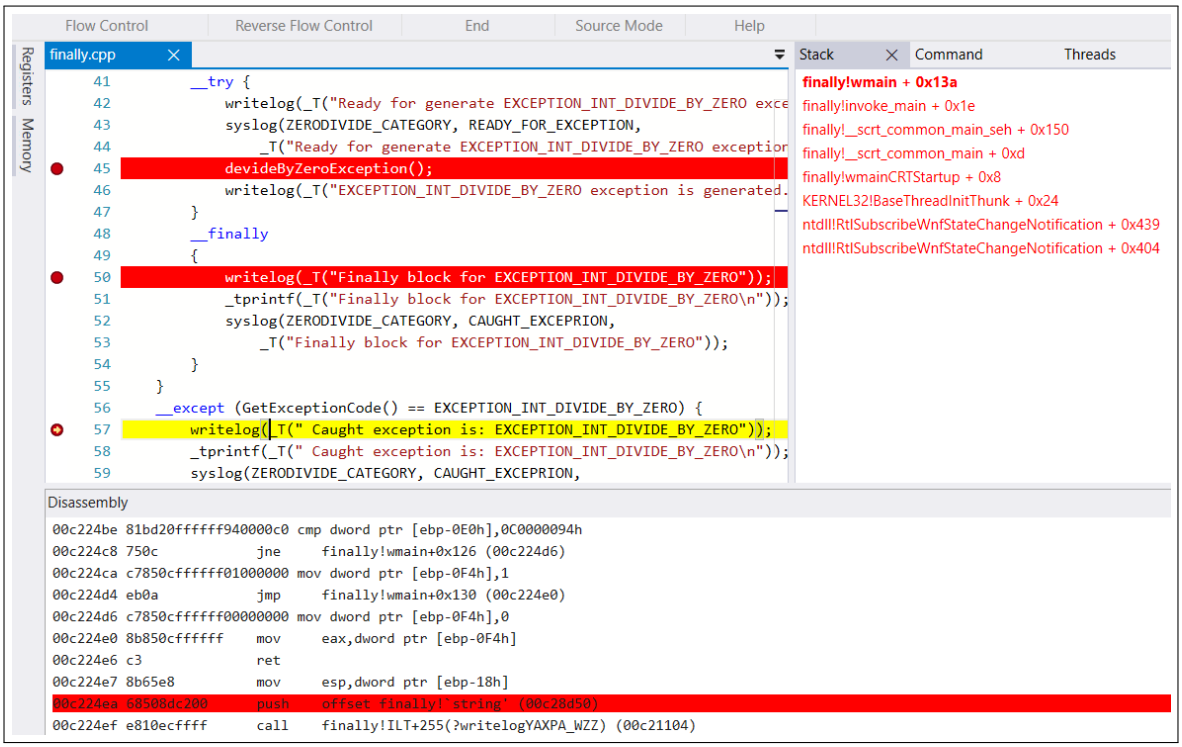



Рис. 12.4: Вызов блока _except

Как и ожидалось, сперва выполнился блок _finally, а затем _except где и было обработано исключение. Стоит отметить то, что было поставлено 3 точки остановки, однако остановок было 4, 4-ой остановкой является сам вызов исключения.

# Использование функции AbnormalTermination

Управление из блока _try может быть передано одним из следующих способов:

- нормальное завершение блока;
- выход из блока при помощи управляющей инструкции leave;
- выход из блока при помощи одной из управляющих инструкций return, break, continue или goto языка программирования C++.

В первых двух случаях считается, что блок try завершился нормально, а в последних двух случаях — ненормально. Для того чтобы определить, как завершился блок try, используется функция AbnormaiTermination, которая имеет следующий прототип:

**BOOL AbnormaiTermination (VOID);**

В случае если блок try завершился ненормально, эта функция возвращает ненулевое значение, а в противном случае — значение false. Используя функцию AbnormaiTermination, ресурсы, захваченные в блоке try, можно освобождать только в том случае, если блок try завершился ненормально. Далее приведена программа, которая использует функцию AbnormaiTermination для проверки нормального завершения блока try.

В далее приведенной программе используются функции goto и leave.

```
#include <stdio.h>
#include <tchar.h>
#include <cstring>
#include <cfloat>
#include <excpt.h>
#include <windows.h>
#include <time.h>

#include "messages.h"

// log
FILE* logfile;
HANDLE eventlog;

void initlog(const _TCHAR* prog);
void closelog();
void writelog(_TCHAR* format, ...);
void syslog(WORD category, WORD identifier, LPWSTR message);


// Defines the entry point for the console application.
int _tmain(int argc, _TCHAR* argv[]) {
    //Init log
```

```
24      initlog(argv[0]);
25      eventlog = RegisterEventSource(NULL, L"MyEventProvider");
26
27      __try {
28          writelog(_T("Call goto"));
29          goto OUT_POINT;
30          writelog(_T("Ready for generate DIVIDE_BY_ZERO exception."));
31          syslog(ZERODIVIDE_CATEGORY, READY_FOR_EXCEPTION,
32              _T("Ready for generate DIVIDE_BY_ZERO exception."));
33          RaiseException(EXCEPTION_INT_DIVIDE_BY_ZERO,
34              EXCEPTION_NONCONTINUABLE, 0, NULL);
35          writelog(_T("DIVIDE_BY_ZERO exception is generated."));
36      }
37      __finally
38      {
39          if (AbnormalTermination()) {
40              writelog(_T("%s"), _T("Abnormal termination in goto case."));
41              _tprintf(_T("%s"), _T("Abnormal termination in goto case.\n"));
42              syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
43                  _T("Abnormal termination in goto case."));
44          }
45          else {
46              writelog(_T("%s"), _T("Normal termination in goto case."));
47              _tprintf(_T("%s"), _T("Normal termination in goto case.\n"));
48              syslog(ZERODIVIDE_CATEGORY, CAUGHT_EXCEPRION,
49                  _T("Normal termination in goto case."));
50          }
51      }
52  OUT_POINT:
53      writelog(_T("A point outside the first __try block."));
54
55      __try {
56          writelog(_T("Call __leave"));
57          __leave;
58          writelog(_T("Ready for generate EXCEPTION_FLT_OVERFLOW exception."));
59          syslog(OVERFLOW_CATEGORY, READY_FOR_EXCEPTION,
60              _T("Ready for generate FLT_OVERFLOW exception."));
61          RaiseException(EXCEPTION_FLT_OVERFLOW,
62              EXCEPTION_NONCONTINUABLE, 0, NULL);
63          writelog(_T("EXCEPTION_FLT_OVERFLOW exception is generated."));
64      }
65      __finally
66      {
67          if (AbnormalTermination()) {
68              writelog(_T("%s"), _T("Abnormal termination in __leave case."));
69              _tprintf(_T("%s"), _T("Abnormal termination in __leave case.\n"));
70              syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
71                  _T("Abnormal termination in __leave case."));
72          }
73          else {
74              writelog(_T("%s"), _T("Normal termination in __leave case."));
75              _tprintf(_T("%s"), _T("Normal termination in __leave case.\n"));
76              syslog(OVERFLOW_CATEGORY, CAUGHT_EXCEPRION,
77                  _T("Normal termination in __leave case."));
78          }
79      }
80      writelog(_T("B point outside the second __try block."));
81
82      closelog();
83      CloseHandle(eventlog);
```

```
84      exit(0);
85  }
86
87  void initlog(const _TCHAR* prog) {
88      _TCHAR logname[255];
89      wcscpy_s(logname, prog);
90
91      // replace extension
92      _TCHAR* extension;
93      extension = wcsstr(logname, _T(".exe"));
94      wcsncpy_s(extension, 5, _T(".log"), 4);
95
96      // Try to open log file for append
97      if (_wfopen_s(&logfile, logname, _T("a+"))) {
98          _wperror(_T("The following error occurred"));
99          _tprintf(_T("Can't open log file %s\n"), logname);
100         exit(1);
101     }
102
103     writelog(_T("%s is starting."), prog);
104 }
105
106 void closelog() {
107     writelog(_T("Shutting down.\n"));
108     fclose(logfile);
109 }
110
111 void writelog(_TCHAR* format, ...) {
112     _TCHAR buf[255];
113     va_list ap;
114
115     struct tm newtime;
116     __time64_t long_time;
117
118     // Get time as 64-bit integer.
119     _time64(&long_time);
120     // Convert to local time.
121     _localtime64_s(&newtime, &long_time);
122
123     // Convert to normal representation.
124     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
125         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
126         newtime.tm_min, newtime.tm_sec);
127
128     // Write date and time
129     fwprintf(logfile, _T("%s"), buf);
130     // Write all params
131     va_start(ap, format);
132     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
133     fwprintf(logfile, _T("%s"), buf);
134     va_end(ap);
135     // New sting
136     fwprintf(logfile, _T("\n"));
137 }
138
139 void syslog(WORD category, WORD identifier, LPWSTR message) {
140     LPWSTR pMessages[1] = { message };
141
142     if (!ReportEvent(
143         eventlog,                   // event log handle
```

```
144             EVENTLOG_INFORMATION_TYPE,  // event type
145             category,                   // event category
146             identifier,                 // event identifier
147             NULL,                       // user security identifier
148             1,                          // number of substitution strings
149             0,                          // data size
150             (LPCWSTR*)pMessages,        // pointer to strings
151             NULL)) {                    // pointer to binary data buffer
152             writelog(_T("ReportEvent failed with 0x%x"), GetLastError());
153         }
154     }
```

Листинг 13.1: finally.cpp

```
1  E:\study\s09\SystemProgramming\reports\Exceptions\Windows\projects\
       ↪ AbnormalTermination\Debug>AbnormalTermination.exe
2  Abnormal termination in goto case.
3  Normal termination in __leave case.
```

Листинг 13.2: Лог консоли при запуске AbnormalTermination.exe

```
1  [8/10/2017 15:56:26] AbnormalTermination.exe is starting.
2  [8/10/2017 15:56:26] Call goto
3  [8/10/2017 15:56:26] Abnormal termination in goto case.
4  [8/10/2017 15:56:26] A point outside the first __try block.
5  [8/10/2017 15:56:26] Call __leave
6  [8/10/2017 15:56:26] Normal termination in __leave case.
7  [8/10/2017 15:56:26] B point outside the second __try block.
8  [8/10/2017 15:56:26] Shutting down.
```

Листинг 13.3: Содержимое файла AbnormalTermination.log

Как и ожидалось AbnormalTermination сигнализировал о ненормальном завершении, при использовании goto, и о нормальном завершении при использовании leave. Работа отладчика не приводится, так как функции goto и leave были рассмотрены ранее.

# Вывод

Механизм структурной обработки исключений в Windows во многом похож на механизма обработки исключений, принятый в языке программирования C++. Так, механизм структурной обработки исключений ориентирован не только на обработку программных исключений, но и на обработку аппаратных исключений. В SEH исключение рассматривается как ошибка, происшедшая при выполнении программы. В языке программирования C++ используется более абстрактный подход и исключение рассматривается как объект произвольного типа, который может выбросить программа, используя оператор throw. Стоит отметить что для удобства работы с этими исключениями, имеется функция-транслятор, которая позволяет преобразовать структурное-исключение в исключение языка C++.

Также стоит отметить полезность программы WinDbg при отладке программы, особенно её последнюю версию, с дружелюбным к пользователю интерфейсом. Используя её имеется возможность просмотреть стек вызовов, диззасемблированный код и многое другое.

# Литература


[1] Настройка WinDbg [Электронный ресурс]. — URL: https://habrahabr.ru/post/187522/ (дата обращения: 2017-10-07).

[2] Quick start to using WinDbg [Электронный ресурс]. — URL: https://www.codeproject.com/Articles/22245/Quick-start-to-using-WinDbg (дата обращения: 2017-10-07).

[3] Регистрация событий C++ приложения в системном журнале Windows [Электронный ресурс]. — URL: https://habrahabr.ru/sandbox/25815/ (дата обращения: 2017-10-06).