

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО  
ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ  
КАФЕДРА КОМПЬЮТЕРНЫХ СИСТЕМ И ПРОГРАММНЫХ ТЕХНОЛОГИЙ

**Отчёт по лабораторной работе №1**  
по курсу «Системное программирование»  
по теме «Обработка исключений в Windows»

Выполнил студент гр. 13541/2:  
Ерниязов Т. Е.

Проверил преподаватель:  
Душутина Е. В.

Санкт-Петербург  
2019 г.

# 1 Цель работы

Познакомится с видами исключений в операционной системе Windows и со способами их обработки.

## 2 Программа работы

1. Сгенерировать и обработать исключения с помощью функций *WinAPI*
2. Получить код исключения с помощью функции *GetExceptionCode*
  - Использовать эту функции в выражении фильтре
  - Использовать эту функцию в обработчике
3. Создать собственную функцию-фильтр
4. Получить информацию об исключении с помощью функции *GetExceptionInformation*
5. Сгенерировать исключение с помощью функции *RaiseException*
6. Использовать функции *UnhandleExceptionFilter* и *Set UnhandleExceptionFilter* для необработанных исключений
7. Вложенная обработка исключений
8. Выйти из блока `__try` с помощью оператора *goto*
9. Выйти из блока `__try` с помощью оператора *leave*
10. Преобразовать структурное исключение в исключение языка C, используя функцию *translator*
11. Использовать финальный обработчик `__finally`
12. Проверить корректность выхода из блока `__try` с помощью функции *AbnormalTermination* в финальном обработчике `__finally`

## 3 Ход работы

### 3.1 Характеристики системы

C:/Users/Lorismelik>systeminfo

Имя узла: TIMUR

Название ОС: Майкрософт Windows 10 Pro

Версия ОС: 10.0.17134 Н/Д построение 17134

Изготовитель ОС: Microsoft Corporation

Параметры ОС: Изолированная рабочая станция

Сборка ОС: Multiprocessor Free

Зарегистрированный владелец: Тимур

Зарегистрированная организация:

Код продукта: 00331-10000-00001-AA048

Дата установки: 28.09.2018, 23:02:07

Время загрузки системы: 09.01.2019, 19:08:35

Изготовитель системы: Acer

Модель системы: Aspire Z5700

Тип системы: x64-based PC

Процессор(ы): Число процессоров - 1. [01]: Intel64 Family 6 Model 30 Stepping 5 GenuineIntel 2801 МГц

Версия BIOS: American Megatrends Inc. P01-A2, 12.03.2010

Папка Windows: C:/Windows

Системная папка: C:/Windows/system32

Устройство загрузки: /Device/HarddiskVolume4

Язык системы: ru;Русский

Язык ввода: ru;Русский

Часовой пояс: (UTC+03:00) Москва, Санкт-Петербург

Полный объем физической памяти: 8 151 МБ

Доступная физическая память: 2 285 МБ

Виртуальная память: Макс. размер: 19 927 МБ  
Виртуальная память: Доступна: 6 794 МБ  
Виртуальная память: Используется: 13 133 МБ  
Расположение файла подкачки: C:/pagefile.sys  
Домен: WORKGROUP  
Сервер входа в сеть: TIMUR

### 3.2 Генерация и обработка исключений

В Windows используется механизм структурной обработки исключений (SEH). В отличие от встроенных средств обработки исключений языка C++, SEH позволяет обрабатывать не только программные исключения, но и аппаратные.

```
1  __try
2  {
3  // secure code
4  }
5  __except (/* exception filter */)
6  {
7  // exception handler
8  }
```

Если при выполнении защищенного кода из блока `__try` возникнет исключение, то ОС перехватит его и приступит к поиску блока `__except`. Найдя его, она передаст управление фильтру исключений. Фильтр исключений может получить код исключения и на основе этого кода принять решение, передать управление обработчику или же сказать системе, чтобы она искала предыдущий по вложенности блок `__except` [1] Фильтр исключений может возвращать одно из трех значений, которые определены в файле `excpt.h`:

- Идентификатор `EXCEPTION_EXECUTE_HANDLER` означает, что для этого блока `__try` есть обработчик исключения и он готов обработать это исключение.
- Идентификатор `EXCEPTION_CONTINUE_SEARCH` означает, что для обработки исключения существует предыдущий по вложенности блок `__except`.
- Идентификатор `EXCEPTION_CONTINUE_EXECUTION` означает, что выполнение продолжится с того участка кода, который вызвал исключение.

Подобная система обработки исключений позволяет организовывать вложенные исключения, что значительно увеличивает гибкость и читабельность языка программирования.

Некоторые типы исключений, которые могут быть обработаны в фильтре:

- `EXCEPTION_ACCESS_VIOLATION` – попытка чтения или записи в виртуальную память без соответствующих прав доступа;
- `EXCEPTION_BREAKPOINT` – встретилась точка останова;
- `EXCEPTION_DATATYPE_MISALIGNMENT` – доступ к данным, адрес которых не выровнен по границе слова или двойного слова;
- `EXCEPTION_SINGLE_STEP` – механизм трассировки программы сообщает, что выполнена одна инструкция;
- `EXCEPTION_ARRAY_BOUNDS_EXCEEDED` – выход за пределы массива, если аппаратное обеспечение поддерживает такую проверку;
- `EXCEPTION_FLT_DENORMAL_OPERAND` – один из операндов с плавающей точкой является ненормализованным;
- `EXCEPTION_FLT_DIVIDE_BY_ZERO` – попытка деления на ноль в операции с плавающей точкой;
- `EXCEPTION_FLT_INEXACT_RESULT` – результат операции с плавающей точкой не может быть точно представлен десятичной дробью;
- `EXCEPTION_FLT_INVALID_OPERATION` – ошибка в операции с плавающей точкой, для которой не предусмотрены другие коды исключения;

- `EXCEPTION_FLT_OVERFLOW` – при выполнении операции с плавающей точкой произошло переполнение;
- `EXCEPTION_FLT_STACK_CHECK` – переполнение или выход за нижнюю границу стека при выполнении операции с плавающей точкой;
- `EXCEPTION_FLT_UNDERFLOW` – результат операции с плавающей точкой является числом, которое меньше минимально возможного числа с плавающей точкой;
- `EXCEPTION_INT_DIVIDE_BY_ZERO` – попытка деления на ноль при операции с целыми числами;
- `EXCEPTION_INT_OVERFLOW` – при выполнении операции с целыми числами произошло переполнение;
- `EXCEPTION_PRIV_INSTRUCTION` – попытка выполнения привилегированной инструкции процессора, которая недопустима в текущем режиме процессора;
- `EXCEPTION_NONCONTINUABLE_EXCEPTION` – попытка возобновления исполнения программы после исключения, которое запрещает выполнять такое действие.

Для примера генерации обработки исключения будут использованы исключения `EXCEPTION_INT_DIVIDE_BY_ZERO` (целочисленное деление на ноль) и `EXCEPTION_DATATYPE_MISALIGNMENT` (доступ к данным, адрес которых не выровнен по границе слова или двойного слова). Первое из них будет брошено системой, из-за попытки выполнения операции деления на ноль, второе, на современных процессорах (x86 и x86-64) исправляется автоматически и не выбрасывается, поэтому его вызов будет имитироваться с помощью `RaiseException`.

Листинг 1: Деление на ноль

```
1 int throw_exception_divide_by_zero()
2 {
3     int one = 1;
4     int zero = 0;
5     return one / zero;
6 }
```

Листинг 2: Попытка доступа к данным с невыровненным адресом

```
1 void throw_exception_datatype_misalignment()
2 {
3     RaiseException(EXCEPTION_DATATYPE_MISALIGNMENT, NULL, NULL, nullptr);
4 }
```

Листинг 3: Обработка ошибок

```
1 int main()
2 {
3     __try
4     {
5         throw_exception_divide_by_zero();
6     }
7     __except (EXCEPTION_EXECUTE_HANDLER)
8     {
9         std::cout << "Caught \"divide by zero\" exception" << std::endl;
10    }
11    __try
12    {
13        throw_exception_datatype_misalignment();
14    }
15    __except (EXCEPTION_EXECUTE_HANDLER)
16    {
17        std::cout << "Caught \"datatype misalignment\" exception" << std::endl;
18    }
19    _getch();
20    return 0;
21 }
```

Функция *main* генерирует исключения и сразу обрабатывает их, используя оператор `__except`, а именно выводит на экран сообщение о пойманном исключении.

```
Caught "divide by zero" exception
Caught "datatype misalignment" exception
Press any key to continue . . . █
```

## Windows Debugger

```

1 ***** Path validation summary *****
2 Response Time (ms) Location
3 eax=00000000 ebx=00000000 ecx=80640000 edx=00000000 esi=009fd000 edi=773cd724
4 eip=774680c9 esp=00b7f2f0 ebp=00b7f31c iopl=0         nv up ei pl zr na pe nc
5 cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
6 ntdll!LdrpDoDebuggerBreak+0x2b:
7 774680c9 cc          int     3
8 Breakpoint 0 hit
9 eax=67d5abe0 ebx=009fd000 ecx=bf68679d edx=67c93a68 esi=00b7f6d0 edi=00b7f7bc
10 eip=00e32801 esp=00b7f6d0 ebp=00b7f7d4 iopl=0         nv up ei pl zr na pe nc
11 cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
12 Project!main+0xe1:
13 00e32801 8bf4          mov     esi,esp
14 0:000> g
15 (1840.1ca4): Integer divide-by-zero - code c0000094 (first chance)
16 First chance exceptions are reported before any exception handling.
17 This exception may be expected and handled.
18 eax=00000002 ebx=009fd000 ecx=bf68679d edx=00000000 esi=00b7f6d0 edi=00b7f7bc
19 eip=00e32838 esp=00b7f6d0 ebp=00b7f7d4 iopl=0         nv up ei pl zr na pe nc
20 cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
21 Project!main+0x118:
22 00e32838 f77dd0          idiv    eax,dword ptr [ebp-30h] ss:002b:00b7f7a4=00000000

```

## Анализ возникшего исключения с помощью команды !analyze -v

```

1 0:000> !analyze -v
2 *****
3                               Exception Analysis
4 *****
5
6 KEY_VALUES_STRING: 1
7
8     Key : Timeline.OS.Boot.DeltaSec
9     Value: 13633
10
11    Key : Timeline.Process.Start.DeltaSec
12    Value: 84
13
14 FAULTING_IP:
15 Project!main+118 [E:\Sys\Project\main.cpp @ 13]
16 00e32838 f77dd0          idiv    eax,dword ptr [ebp-30h]
17 EXCEPTION_RECORD: (.exr -1)
18 ExceptionAddress: 00e32838 (Project!main+0x00000118)
19 ExceptionCode: c0000094 (Integer divide-by-zero)
20 ExceptionFlags: 00000000
21 NumberParameters: 0
22 FAULTING_THREAD: 00001ca4
23 DEFAULT_BUCKET_ID: INTEGER_DIVIDE_BY_ZERO
24 PROCESS_NAME: Project.exe
25 ERROR_CODE: (NTSTATUS) 0xc0000094 - <Unable to get error code text>
26 EXCEPTION_CODE: (NTSTATUS) 0xc0000094 - <Unable to get error code text>
27 EXCEPTION_CODE_STR: c0000094
28 BUGCHECK_STR: INTEGER_DIVIDE_BY_ZERO
29 PRIMARY_PROBLEM_CLASS: INTEGER_DIVIDE_BY_ZERO
30 PROBLEM_CLASSES:
31
32     ID: [On321]
33     Type: [APPLICATION_FAULT_STRING]
34     Class: Primary
35     Scope: DEFAULT_BUCKET_ID (Failure Bucket ID prefix)
36           BUCKET_ID
37     Name: Omit
38     Data: Add
39           String: [INTEGER_DIVIDE_BY_ZERO]
40     PID: [Unspecified]
41     TID: [Unspecified]
42     Frame: [0]
43
44 LAST_CONTROL_TRANSFER: from 00e3320e to 00e32838
45 STACK_TEXT:
46 00b7f7d4 00e3320e 00000001 00bf6400 00bfca60 Project!main+0x118
47 00b7f7e8 00e33077 d84b3fd1 00e313cf 00e313cf Project!invoke_main+0x1e
48 00b7f844 00e32f0d 00b7f854 00e33288 00b7f868 Project!__scrt_common_main_seh+0x157
49 00b7f84c 00e33288 00b7f868 76f58484 009fd000 Project!__scrt_common_main+0xd
50 00b7f854 76f58484 009fd000 76f58460 ae57d471 Project!mainCRTStartup+0x8
51 00b7f868 7742305a 009fd000 afe3de48 00000000 KERNEL32!BaseThreadInitThunk+0x24
52 00b7f8b0 7742302a ffffffff 7743ecb7 00000000 ntdll!_RtlUserThreadStart+0x2f
53 00b7f8c0 00000000 00e313cf 009fd000 00000000 ntdll!_RtlUserThreadStart+0x1b
54 STACK_COMMAND: ~0s ; .cxr ; kb
55 FAULT_INSTR_CODE: 89d07df7
56 FAULTING_SOURCE_LINE_NUMBER: 13
57 FAULTING_SOURCE_CODE:
58     11: int one = 1;
59     12: int zero = 0;
60 > 13: return one / zero;
61     14: }
62     15:
63 FAILURE_EXCEPTION_CODE: c0000094
64 BUCKET_ID_PREFIX_STR: INTEGER_DIVIDE_BY_ZERO_

```

```

65 FAILURE_PROBLEM_CLASS:  INTEGER_DIVIDE_BY_ZERO
66 eax=00000000 ebx=00000000 ecx=00000000 edx=00000000 esi=00000000 edi=774d79a0
67 eip=7742a52c esp=00b7f6bc ebp=00b7f794 iopl=0         nv up ei pl nz na po nc
68 cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
69 ntdll!NtTerminateProcess+0xc:
70 7742a52c c20800          ret     8

```

### 3.3 Получение кода исключения

Для того, чтобы получить какую-либо информацию об обрабатываемом исключении можно использовать функцию *GetExceptionCode()*, которая возвращает код полученного исключения. Функция *GetExceptionCode* может вызываться только в выражении-фильтре или в блоке обработки исключения. Следовательно, эта функция вызывается всегда только в том случае, если исключение произошло. Отсюда можно определить назначение функции *GetExceptionCode*. Если эта функция вызывается в выражении фильтра, то она используется для того, чтобы определить выполняет ли текущий обработчик исключения обработку исключений с данным кодом или нужно продолжить поиск подходящего обработчика исключения. Если же функция *GetExceptionCode* вызывается в блоке обработки исключения, то она также предназначена для проверки кодов исключений, которые обрабатывает текущий обработчик исключения, но в этом случае поиск другого обработчика исключений не выполняется.

```

1 void handle_exception(DWORD code)
2 {
3     std::cout << "Caught exception with code = " << code << std::endl;
4     switch (code)
5     {
6     case EXCEPTION_INT_DIVIDE_BY_ZERO:
7         std::cout << "EXCEPTION_INT_DIVIDE_BY_ZERO" << std::endl;
8         break;
9     case EXCEPTION_DATATYPE_MISALIGNMENT:
10        std::cout << "EXCEPTION_DATATYPE_MISALIGNMENT" << std::endl;
11        break;
12    default:
13        std::cout << "unrecognized exception" << std::endl;
14        break;
15    }
16 }

```

```

1 int main()
2 {
3     __try
4     {
5         throw_exception_divide_by_zero();
6     }
7     __except (EXCEPTION_EXECUTE_HANDLER)
8     {
9         handle_exception(GetExceptionCode());
10    }
11    __try
12    {
13        throw_exception_datatype_misalignment();
14    }
15    __except (EXCEPTION_EXECUTE_HANDLER)
16    {
17        handle_exception(GetExceptionCode());
18    }
19    _getch();
20    return 0;
21 }

```

```

Caught exception with code = 3221225620
EXCEPTION_INT_DIVIDE_BY_ZERO
Caught exception with code = 2147483650
EXCEPTION_DATATYPE_MISALIGNMENT
Press any key to continue . . . █

```

Рис. 1: Вывод программы

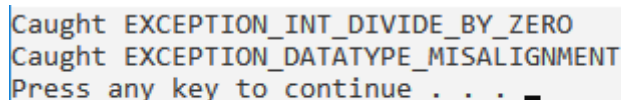
Так же можно использовать данную функцию в выражении-фильтре:

```

1 LONG filter(DWORD actual_code, DWORD expected_code)
2 {
3     if (actual_code == expected_code)
4     {
5         return EXCEPTION_EXECUTE_HANDLER;
6     }
7     else
8     {
9         return EXCEPTION_CONTINUE_SEARCH;
10    }
11 }

1 int main()
2 {
3     __try
4     {
5         throw_exception_divide_by_zero();
6     }
7     __except (filter(GetExceptionCode(), EXCEPTION_INT_DIVIDE_BY_ZERO))
8     {
9         std::cout << "Caught EXCEPTION_INT_DIVIDE_BY_ZERO" << std::endl;
10    }
11    __try
12    {
13        throw_exception_datatype_misalignment();
14    }
15    __except (filter(GetExceptionCode(), EXCEPTION_DATATYPE_MISALIGNMENT))
16    {
17        std::cout << "Caught EXCEPTION_DATATYPE_MISALIGNMENT" << std::endl;
18    }
19    _getch();
20    return 0;
21 }

```



```

Caught EXCEPTION_INT_DIVIDE_BY_ZERO
Caught EXCEPTION_DATATYPE_MISALIGNMENT
Press any key to continue . . . 

```

Рис. 2: Вывод программы

## Windows Debugger

```

1 ***** Path validation summary *****
2 Response Time (ms) Location
3 eax=00000000 ebx=00000000 ecx=3c6c0000 edx=00000000 esi=00f22000 edi=7779d6b4
4 eip=77838679 esp=010ff380 ebp=010ff3ac iopl=0 nv up ei pl zr na pe nc
5 cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000246
6 ntdll!LdrpDoDebuggerBreak+0x2b:
7 77838679 cc int 3
8 (1a5c.1f94): Integer divide-by-zero - code c0000094 (first chance)
9 First chance exceptions are reported before any exception handling.
10 This exception may be expected and handled.
11 eax=00000006 ebx=00f22000 ecx=1e8bd75b edx=00000000 esi=010ff754 edi=010ff84c
12 eip=00b62838 esp=010ff754 ebp=010ff864 iopl=0 nv up ei pl zr na pe nc
13 cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00010246
14 Project!main+0x118:
15 00b62838 f77dd0 idiv eax,dword ptr [ebp-30h] ss:002b:010ff834=00000000
16 0:000> !analyze -v
17 *****
18 * Exception Analysis *
19 *****
20
21 KEY_VALUES_STRING: 1
22
23 Key : Timeline.OS.Boot.DeltaSec
24 Value: 827
25
26 Key : Timeline.Process.Start.DeltaSec
27 Value: 41
28 FAULTING_IP:
29 Project!main+118 E:\Sys\Project\main.cpp programming\lab1\Project1\main.cpp @ 13]
30 00b62838 f77dd0 idiv eax,dword ptr [ebp-30h]
31
32 EXCEPTION_RECORD: (.exr -1)
33 ExceptionAddress: 00b62838 (Project!main+0x00000118)
34 ExceptionCode: c0000094 (Integer divide-by-zero)
35 ExceptionFlags: 00000000
36 NumberParameters: 0
37
38 FAULTING_THREAD: 00001f94
39 DEFAULT_BUCKET_ID: INTEGER_DIVIDE_BY_ZERO

```

```

40 | PROCESS_NAME: Project.exe
41 | ERROR_CODE: (NTSTATUS) 0xc0000094 - <Unable to get error code text>
42 | EXCEPTION_CODE: (NTSTATUS) 0xc0000094 - <Unable to get error code text>
43 | EXCEPTION_CODE_STR: c0000094
44 | BUGCHECK_STR: INTEGER_DIVIDE_BY_ZERO
45 | PRIMARY_PROBLEM_CLASS: INTEGER_DIVIDE_BY_ZERO
46 | PROBLEM_CLASSES:
47 |
48 |     ID: [0n321]
49 |     Type: [APPLICATION_FAULT_STRING]
50 |     Class: Primary
51 |     Scope: DEFAULT_BUCKET_ID (Failure Bucket ID prefix)
52 |     BUCKET_ID
53 |     Name: Omit
54 |     Data: Add
55 |     String: [INTEGER_DIVIDE_BY_ZERO]
56 |     PID: [Unspecified]
57 |     TID: [Unspecified]
58 |     Frame: [0]
59 | LAST_CONTROL_TRANSFER: from 00b6322e to 00b62838
60 | STACK_TEXT:
61 | 010ff864 00b6322e 00000001 013346d0 01334878 Project2!main+0x118
62 | 010ff878 00b63097 7e8f510d 00b613cf 00b613cf Project2!_scrt_narrow_environment_policy::
        initialize_environment+0x2e
63 | 010ff8d4 00b62f2d 010ff8e4 00b632a8 010ff8f8 Project2!_except_handler4+0x2b7
64 | 010ff8dc 00b632a8 010ff8f8 740f8484 00f22000 Project2!_except_handler4+0x14d
65 | 010ff8e4 740f8484 00f22000 740f8460 0a20975a Project!mainCRTStartup+0x8
66 | 010ff8f8 777f2ec0 00f22000 09b36137 00000000 KERNEL32!BaseThreadInitThunk+0x24
67 | 010ff940 777f2e90 ffffffff 7780debd 00000000 ntdll!_RtlUserThreadStart+0x2f
68 | 010ff950 00000000 00b613cf 00f22000 00000000 ntdll!_RtlUserThreadStart+0x1b
69 | STACK_COMMAND: ~Os ; .cxr ; kb
70 | FAULT_INSTR_CODE: 89d07df7
71 | FAULTING_SOURCE_LINE:
72 | E:\Sys\Project\main.cpp
73 | FAULTING_SOURCE_FILE:
74 | E:\Sys\Project\main.cpp
75 | FAULTING_SOURCE_LINE_NUMBER: 13
76 | FAULTING_SOURCE_CODE:
77 | 11: int one = 1;
78 | 12: int zero = 0;
79 | > 13: return one / zero;
80 | 14: }
81 | 15:
82 | FAILURE_EXCEPTION_CODE: c0000094
83 | SYMBOL_STACK_INDEX: 0
84 | BUCKET_ID: INTEGER_DIVIDE_BY_ZERO_Project2!main+118
85 | FAILURE_EXCEPTION_CODE: c0000094
86 | BUCKET_ID_PREFIX_STR: INTEGER_DIVIDE_BY_ZERO_
87 | FAILURE_PROBLEM_CLASS: INTEGER_DIVIDE_BY_ZERO

```

### 3.4 Получение информации об исключении с помощью GetExceptionInformation

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

```
1 | LPEXCEPTION_POINTERS GetExceptionInformation(VOID);
```

Эта функция возвращает указатель на структуру типа:

```

1 | typedef struct EXCEPTION_POINTERS {
2 |     PEXCEPTION_RECORD Except ionRecord;
3 |     PCONTEXT Context;
4 | } EXCEPTION_POINTERS, * PEXCEPTION_POINTERS;

```

которая, в свою очередь, содержит два указателя: ExceptionRecord и context на структуры типа exception\_record и context соответственно.

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

Структура типа exception\_record имеет следующий формат:

```

1 | typedef struct EXCEPTION_RECORD {
2 |     DWORD Except i onCode;
3 |     DWORD ExceptionFlags,
4 |     strict _EXCEPTION_RECORD *ExceptionRecord;
5 |     PVOID ExceptionAddress,
6 |     DWORD Number Parameters,
7 |     ULONG_PTR ExceptionInf ormation [ EXCEPTION_MAXIMUM_PARAMETERS ] ;
8 | } EXCEPTION_RECORD, *PEXCEPTION_RECORD;

```

В нее система записывает информацию об исключении. Поля этой структуры имеют следующее назначение. Поле ExceptionCode содержит код исключения, который может принимать такие же значения, как



и код исключения, возвращаемый функцией `GetExceptionCode`. Поле `ExceptionFlags` может принимать одно из двух значений:

- 0 — которое обозначает, что после обработки исключения возможно возобновление выполнения программы;
- `EXCEPTION_NONCONTINUABLE` — которое обозначает, что после обработки исключения возобновление выполнения программы невозможно.

Если установлено значение `EXCEPTION_NONCONTINUABLE` и выполнена попытка возобновления выполнения программы, то система выбросит исключение `EXCEPTION_NONCONTINUABLE_EXCEPTION`.

Поле `ExceptionRecord` содержит указатель на следующую структуру типа `exception_record`, которая может быть создана в случае вложенных исключений.

Поле `ExceptionAddress` содержит адрес инструкции в программе, на которой произошло исключение.

Поле `NumberParameters` содержит количество параметров, заданных в поле `ExceptionInformation`, которое является последним в этой структуре.

Поле `ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS]` Определяет массив 32-битных аргументов, которые описывают исключение. Элементы этого массива могут использоваться функцией генерации программных исключений `RaiseException`.

Сделаем важное замечание о том, что функция `GetExceptionInformation` может вызываться только в выражении фильтра. Поэтому эта функция вызывается всегда только в том случае, если исключение произошло. Кроме того, структуры типа `exception_pointers`, `exception_record` и `context` действительны только на время вычисления выражения-фильтра. Чтобы использовать содержимое структур типа `exception_record` и `context` в блоке обработки исключения, его нужно сохранить в объявленных в программе переменных такого же типа. Как видно из описания структуры `EXCEPTION_RECORD`, функцию `GetExceptionInformation` можно использовать для двух целей: первая цель заключается в получении более подробной информации об исключении, учитывая содержимое структуры типа `context`; вторая цель состоит в обработке вложенных исключений.

```
1 EXCEPTION_RECORD er;
2
3 LONG filter(DWORD actualCode, DWORD expectedCode, EXCEPTION_POINTERS* exceptionPointers)
4 {
5     er = *(exceptionPointers->ExceptionRecord);
6     return filter(actualCode, expectedCode);
7 }
8
9 void show_info()
10 {
11     std::cout << "code: " << er.ExceptionCode << std::endl;
12     std::cout << "flags: " << er.ExceptionFlags << std::endl;
13     std::cout << "address: " << er.ExceptionAddress << std::endl;
14     std::cout << "record: " << er.ExceptionRecord << std::endl;
15     std::cout << "NumberParameters: " << er.NumberParameters << std::endl;
16 }

1 int main()
2 {
3     __try
4     {
5         throw_exception_divide_by_zero();
6     }
7     __except (filter(GetExceptionCode(), EXCEPTION_INT_DIVIDE_BY_ZERO, GetExceptionInformation()))
8     {
9         std::cout << "Caught \"divide by zero\" exception\n" << std::endl;
10        show_info();
11    }
12    std::cout << "\n\n" << std::endl;
13    __try
14    {
15        throw_exception_datatype_misalignment();
16    }
17    __except (filter(GetExceptionCode(), EXCEPTION_DATATYPE_MISALIGNMENT, GetExceptionInformation()))
18    {
19        std::cout << "Caught \"datatype misalignment\" exception\n" << std::endl;
20        show_info();
21    }
22    _getch();
23    return 0;
24 }
```

В результате выполнения программы получаем:

```

Caught "divide by zero" exception

code: 3221225620
flags: 0
address: 00007FF74C0A281C
record: 0000000000000000
NumberParameters: 0

Caught "datatype misalignment" exception

code: 2147483650
flags: 0
address: 00007FF8B9FCA388
record: 0000000000000000
NumberParameters: 0

```

Рис. 3: Вывод программы

### 3.5 Программная генерация исключения RaiseException

Для программной генерации исключений можно использовать функцию `RaiseException()`. Данная функция принимает 4 аргумента:

- `dwExceptionCode` - код исключения
- `dwExceptionFlags` - флаг возобновляемого исключения
- `nNumberOfArguments` - количество аргументов
- `lpArguments` - массив аргументов

Листинг 4: Прототип `RaiseException`

```

1 WINBASEAPI
2 __analysis_noreturn
3 VOID
4 WINAPI
5 RaiseException(
6     _In_ DWORD dwExceptionCode,
7     _In_ DWORD dwExceptionFlags,
8     _In_ DWORD nNumberOfArguments,
9     _In_reads_opt_(nNumberOfArguments) CONST ULONG_PTR* lpArguments
10 );

1 LONG filter(EXCEPTION_POINTERS* exceptionPointers) {
2     er = *(exceptionPointers->ExceptionRecord);
3     return EXCEPTION_EXECUTE_HANDLER;
4 }

1 int main()
2 {
3     int a = 3;
4     int b = 5;
5     DWORD Arguments[2];
6     __try
7     {
8         Arguments[0] = a;
9         Arguments[1] = b;
10        RaiseException(0xFF, 0, 2, Arguments);
11    }
12    __except (filter(GetExceptionInformation()))
13    {
14        std::cout << "Exception code: " << std::hex << er.ExceptionCode << std::endl;
15        std::cout << "Number parameters: " << er.NumberParameters << std::endl;
16        std::cout << "1 parameter: " << std::dec << er.ExceptionInformation[0] << std::endl;
17        std::cout << "2 parameter: " << er.ExceptionInformation[1] << std::endl;
18    }
19    _getch();
20    return 0;
21 }

```

В результате программы получаем:

```
Exception code: ff
Number parameters: 2
1 parameter: 3
2 parameter: 5
```

Рис. 4: Вывод программы

### 3.6 Обработка необработанных исключений

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:

Листинг 5: Сигнатура `UnhandledExceptionFilter`

```
1 LONG UnhandledExceptionFilter(PEXCEPTION_POINTERS pExceptionInfo);
```

Эта функция имеет один параметр, который указывает на структуру с типом `exception_info` и возвращает одно из следующих значений:

- `EXCEPTION_CONTINUE_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:

Листинг 6: Сигнатуры `SetUnhandledExceptionFilter`

```
1 WINBASEAPI
2 LPTOP_LEVEL_EXCEPTION_FILTER
3 WINAPI
4 SetUnhandledExceptionFilter(_In_opt_ 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`

```
1 LONG filterWithPrint(PEXCEPTION_POINTERS exceptionPointers)
2 {
3     er = *(exceptionPointers->ExceptionRecord);
4     show_info();
5     system("pause");
6     return EXCEPTION_EXECUTE_HANDLER;
7 }
```

```
1 int main()
2 {
3     auto old_filter = SetUnhandledExceptionFilter((LPTOP_LEVEL_EXCEPTION_FILTER)filterWithPrint);
4     throw_exception_datatype_misalignment();
5     std::cout << "After exception" << std::endl;
6     system("pause");
7     return 0;
8 }
```

В результате выполнения программы:

```
code: 2147483650  
flags: 0  
address: 00007FF8B9FCA388  
record: 0000000000000000  
NumberParameters: 0  
Press any key to continue . . .
```

Рис. 5: Вывод программы

Продemonстрируем работу Process Monitor и покажем вызовы на стеке. Запустив утилиту, будет выдано большое количество процессов. Выбрав нашу программу и добавив её в основное дерево процессов утилиты, можно увидеть стек вызовов в окне <Stack Summary>:

Name	Count	% Count	Time	% Time	Location	Module
U <All>	1553	100.00000%	0.5862349	100.00000%	<All>	<All>
U Conhost.exe	902	58.08113%	0.0205509	3.50557%	Conhost.exe	Conhost.exe(16448)
U Project.exe	651	41.91887%	0.5656840	96.49443%	Project.exe	Project.exe(16776)
U <unknown> + 0x1c58b390124	524	33.74115%	0.0381338	6.50487%	<unknown> + 0x1...	<unknown> 0x1c
U <unknown> + 0x7fa0dd71410	1	0.06439%	0.0000000	0.00000%	<unknown> + 0x7...	<unknown> 0x7f
U <unknown> + 0x7fa0dd71431	2	0.12878%	0.0000000	0.00000%	<unknown> + 0x7...	<unknown> 0x7f
K KeSynchronizeExecution + 0x63e6	1	0.06439%	0.0000000	0.00000%	KeSynchronizeEx...	ntoskrnl.exe
U LdrInitializeThunk + 0xe	77	4.95815%	0.0100413	1.71265%	LdrInitializeThun...	ntdll.dll
U RtlUserThreadStart	4	0.25757%	0.4942341	84.30650%	RtlUserThreadSta...	ntdll.dll
U RtlUserThreadStart + 0x21	41	2.64005%	0.0232500	3.96599%	RtlUserThreadSta...	ntdll.dll
U BaseThreadInitThunk + 0x14	41	2.64005%	0.0232500	3.96599%	BaseThreadInitTh...	kernel32.dll
U mainCRTStartup + 0x9	2	0.12878%	0.000104	0.00177%	mainCRTStartup +...	Project.exe
U _scrt_common_main + 0xe	2	0.12878%	0.000104	0.00177%	_scrt_common_...	Project.exe
U _scrt_common_main_seh + 0x12e	1	0.06439%	0.000049	0.00084%	_scrt_common_...	Project.exe
U _scrt_common_main_seh + 0x89	1	0.06439%	0.000055	0.00094%	_scrt_common_...	Project.exe
U InitTerm + 0x59	1	0.06439%	0.000055	0.00094%	InitTerm + 0x59	ucrtbased.dll
U pre_cxx_initialization + 0x9	1	0.06439%	0.000055	0.00094%	pre_cxx_initia...	Project.exe
U _scrt_set_unhandled_exception_filter + 0x11	1	0.06439%	0.000055	0.00094%	_scrt_set_unhan...	Project.exe
U SetUnhandledExceptionFilter + 0x28	1	0.06439%	0.000055	0.00094%	SetUnhandledExc...	KernelBase.dll
U SetUnhandledExceptionFilter + 0x252	1	0.06439%	0.000055	0.00094%	SetUnhandledExc...	KernelBase.dll
U NtQueryVirtualMemory + 0x14	1	0.06439%	0.000055	0.00094%	NtQueryVirtualMe...	ntdll.dll
K setjmpex + 0x6ea3	1	0.06439%	0.000055	0.00094%	setjmpex + 0x6ea3	ntoskrnl.exe
K NtAllocateVirtualMemory + 0x1191	1	0.06439%	0.000055	0.00094%	NtAllocateVirtual...	ntoskrnl.exe
K NtAllocateVirtualMemory + 0x1a31	1	0.06439%	0.000055	0.00094%	NtAllocateVirtual...	ntoskrnl.exe
K SeQuerySecurityDescriptorInfo + 0x20c5	1	0.06439%	0.000055	0.00094%	SeQuerySecurityD...	ntoskrnl.exe
K CmCallbackGetObjectIDEx + 0x786	1	0.06439%	0.000055	0.00094%	CmCallbackGetKe...	ntoskrnl.exe
K CmCallbackGetObjectIDEx + 0x4c4	1	0.06439%	0.000055	0.00094%	CmCallbackGetKe...	ntoskrnl.exe
K MmCopyVirtualMemory + 0xa57	1	0.06439%	0.000055	0.00094%	MmCopyVirtualMe...	ntoskrnl.exe
K IoCallDriver + 0x59	1	0.06439%	0.000055	0.00094%	IoCallDriver + 0...	ntoskrnl.exe
K RtlDecodeParameters + 0x3ee	1	0.06439%	0.000055	0.00094%	RtlDecodeParame...	FLTMGR.SYS
K RtlDecodeParameters + 0x5f4	1	0.06439%	0.000055	0.00094%	RtlDecodeParame...	FLTMGR.SYS
K RtlDecodeParameters + 0x14dc	1	0.06439%	0.000055	0.00094%	RtlDecodeParame...	FLTMGR.SYS
K RtlDecodeParameters + 0x193c	1	0.06439%	0.000055	0.00094%	RtlDecodeParame...	FLTMGR.SYS
U RtlReleaseSRWLockExclusive + 0x739	39	2.51127%	0.0232396	3.96421%	RtlReleaseSRWL...	ntdll.dll
U RtlAcquireSRWLockExclusive + 0x338	2	0.12878%	0.0000000	0.00000%	RtlAcquireSRWL...	ntdll.dll
U RtlAcquireSRWLockExclusive + 0x482	37	2.38249%	0.0232396	3.96421%	RtlAcquireSRWL...	ntdll.dll
K setjmpex + 0x6ea3	1	0.06439%	0.0000248	0.00423%	setjmpex + 0x6ea3	ntoskrnl.exe

Рис. 6: Стек вызовов

Также продемонстрируем работу отладчика OllyDbg. С помощью данного отладчика можно проанализировать работу программы. Можно пошагово исполнить программу, поставить точки останова и отслеживать всю интересующую информацию.

Address	Disassembly	Comment
00401000	JMP EBX	
00401001	JMP EBX	
00401002	JMP EBX	
00401003	JMP EBX	
00401004	JMP EBX	
00401005	JMP EBX	
00401006	JMP EBX	
00401007	JMP EBX	
00401008	JMP EBX	
00401009	JMP EBX	
0040100A	JMP EBX	
0040100B	JMP EBX	
0040100C	JMP EBX	
0040100D	JMP EBX	
0040100E	JMP EBX	
0040100F	JMP EBX	
00401010	JMP EBX	
00401011	JMP EBX	
00401012	JMP EBX	
00401013	JMP EBX	
00401014	JMP EBX	
00401015	JMP EBX	
00401016	JMP EBX	
00401017	JMP EBX	
00401018	JMP EBX	
00401019	JMP EBX	
0040101A	JMP EBX	
0040101B	JMP EBX	
0040101C	JMP EBX	
0040101D	JMP EBX	
0040101E	JMP EBX	
0040101F	JMP EBX	
00401020	JMP EBX	
00401021	JMP EBX	
00401022	JMP EBX	
00401023	JMP EBX	
00401024	JMP EBX	
00401025	JMP EBX	
00401026	JMP EBX	
00401027	JMP EBX	
00401028	JMP EBX	
00401029	JMP EBX	
0040102A	JMP EBX	
0040102B	JMP EBX	
0040102C	JMP EBX	
0040102D	JMP EBX	
0040102E	JMP EBX	
0040102F	JMP EBX	
00401030	JMP EBX	
00401031	JMP EBX	
00401032	JMP EBX	
00401033	JMP EBX	
00401034	JMP EBX	
00401035	JMP EBX	
00401036	JMP EBX	
00401037	JMP EBX	
00401038	JMP EBX	
00401039	JMP EBX	
0040103A	JMP EBX	
0040103B	JMP EBX	
0040103C	JMP EBX	
0040103D	JMP EBX	
0040103E	JMP EBX	
0040103F	JMP EBX	
00401040	JMP EBX	
00401041	JMP EBX	
00401042	JMP EBX	
00401043	JMP EBX	
00401044	JMP EBX	
00401045	JMP EBX	
00401046	JMP EBX	
00401047	JMP EBX	
00401048	JMP EBX	
00401049	JMP EBX	
0040104A	JMP EBX	
0040104B	JMP EBX	
0040104C	JMP EBX	
0040104D	JMP EBX	
0040104E	JMP EBX	
0040104F	JMP EBX	
00401050	JMP EBX	
00401051	JMP EBX	
00401052	JMP EBX	
00401053	JMP EBX	
00401054	JMP EBX	
00401055	JMP EBX	
00401056	JMP EBX	
00401057	JMP EBX	
00401058	JMP EBX	
00401059	JMP EBX	
0040105A	JMP EBX	
0040105B	JMP EBX	
0040105C	JMP EBX	
0040105D	JMP EBX	
0040105E	JMP EBX	
0040105F	JMP EBX	
00401060	JMP EBX	
00401061	JMP EBX	
00401062	JMP EBX	
00401063	JMP EBX	
00401064	JMP EBX	
00401065	JMP EBX	
00401066	JMP EBX	
00401067	JMP EBX	
00401068	JMP EBX	
00401069	JMP EBX	
0040106A	JMP EBX	
0040106B	JMP EBX	
0040106C	JMP EBX	
0040106D	JMP EBX	
0040106E	JMP EBX	
0040106F	JMP EBX	
00401070	JMP EBX	
00401071	JMP EBX	
00401072	JMP EBX	
00401073	JMP EBX	
00401074	JMP EBX	
00401075	JMP EBX	
00401076	JMP EBX	
00401077	JMP EBX	
00401078	JMP EBX	
00401079	JMP EBX	
0040107A	JMP EBX	
0040107B	JMP EBX	
0040107C	JMP EBX	
0040107D	JMP EBX	
0040107E	JMP EBX	
0040107F	JMP EBX	
00401080	JMP EBX	
00401081	JMP EBX	
00401082	JMP EBX	
00401083	JMP EBX	
00401084	JMP EBX	
00401085	JMP EBX	
00401086	JMP EBX	
00401087	JMP EBX	
00401088	JMP EBX	
00401089	JMP EBX	
0040108A	JMP EBX	
0040108B	JMP EBX	
0040108C	JMP EBX	
0040108D	JMP EBX	
0040108E	JMP EBX	
0040108F	JMP EBX	
00401090	JMP EBX	
00401091	JMP EBX	
00401092	JMP EBX	
00401093	JMP EBX	
00401094	JMP EBX	
00401095	JMP EBX	
00401096	JMP EBX	
00401097	JMP EBX	
00401098	JMP EBX	
00401099	JMP EBX	
0040109A	JMP EBX	
0040109B	JMP EBX	
0040109C	JMP EBX	
0040109D	JMP EBX	
0040109E	JMP EBX	
0040109F	JMP EBX	
004010A0	JMP EBX	
004010A1	JMP EBX	
004010A2	JMP EBX	
004010A3	JMP EBX	
004010A4	JMP EBX	
004010A5	JMP EBX	
004010A6	JMP EBX	
004010A7	JMP EBX	
004010A8	JMP EBX	
004010A9	JMP EBX	
004010AA	JMP EBX	
004010AB	JMP EBX	
004010AC	JMP EBX	
004010AD	JMP EBX	
004010AE	JMP EBX	
004010AF	JMP EBX	
004010B0	JMP EBX	
004010B1	JMP EBX	
004010B2	JMP EBX	
004010B3	JMP EBX	
004010B4	JMP EBX	
004010B5	JMP EBX	
004010B6	JMP EBX	
004010B7	JMP EBX	
004010B8	JMP EBX	
004010B9	JMP EBX	
004010BA	JMP EBX	
004010BB	JMP EBX	
004010BC	JMP EBX	
004010BD	JMP EBX	
004010BE	JMP EBX	
004010BF	JMP EBX	
004010C0	JMP EBX	
004010C1	JMP EBX	
004010C2	JMP EBX	
004010C3	JMP EBX	
004010C4	JMP EBX	
004010C5	JMP EBX	
004010C6	JMP EBX	
004010C7	JMP EBX	
004010C8	JMP EBX	
004010C9	JMP EBX	
004010CA	JMP EBX	
004010CB	JMP EBX	
004010CC	JMP EBX	
004010CD	JMP EBX	
004010CE	JMP EBX	
004010CF	JMP EBX	
004010D0	JMP EBX	
004010D1	JMP EBX	
004010D2	JMP EBX	
004010D3	JMP EBX	
004010D4	JMP EBX	
004010D5	JMP EBX	
004010D6	JMP EBX	
004010D7	JMP EBX	
004010D8	JMP EBX	
004010D9	JMP EBX	
004010DA	JMP EBX	
004010DB	JMP EBX	
004010DC	JMP EBX	
004010DD	JMP EBX	
004010DE	JMP EBX	
004010DF	JMP EBX	
004010E0	JMP EBX	
004010E1	JMP EBX	
004010E2	JMP EBX	
004010E3	JMP EBX	
004010E4	JMP EBX	
004010E5	JMP EBX	
004010E6	JMP EBX	
004010E7	JMP EBX	
004010E8	JMP EBX	
004010E9	JMP EBX	
004010EA	JMP EBX	
004010EB	JMP EBX	
004010EC	JMP EBX	
004010ED	JMP EBX	
004010EE	JMP EBX	</

```

2 {
3   __try
4   {
5     __try
6     {
7       throw_exception_divide_by_zero();
8     }
9     __except (filter(GetExceptionCode(), EXCEPTION_DATATYPE_MISALIGNMENT))
10    {
11      std::cout << "Caught \"datatype misalignment\" exception" << std::endl;
12    }
13  }
14  __except (filter(GetExceptionCode(), EXCEPTION_INT_DIVIDE_BY_ZERO))
15  {
16    std::cout << "Caught \"divide by zero\" exception" << std::endl;
17  }
18  system("pause");
19  return 0;
20 }

```

В результате выполнения программы:

```

Caught "divide by zero" exception
Press any key to continue . . .

```

Рис. 8: Вывод программы

### 3.8 Завершение блока \_\_try, используя goto и \_\_leave

Для передачи управления из фрейма можно использовать инструкцию goto из C++. В этом случае система считает, что блок с охраняемым кодом завершился аварийно и поэтому выполняет глобальное раскручивание стека. Следовательно, использование инструкции goto вызывает исполнение дополнительного программного кода, что замедляет выполнение программы.

Программа, использующая goto для выхода из блока \_\_try.

```

1 int main()
2 {
3   __try
4   {
5     goto label;
6     throw_exception_divide_by_zero();
7   }
8   __finally
9   {
10    printf("__finally \n");
11  }
12 label:
13   system("pause");
14   return 0;
15 }

```

Её ассемблерный код приведен ниже. Из этого кода видно, что на месте вызова goto, стоит вызов функции \_local\_unwind, которая выполняет раскрутку стека и переводит исполнение программы к маркеру \$label\$12.

Листинг 7: Скомпилированный ассемблерный код

```

1 $T1 = 32
2 argc$ = 64
3 argv$ = 72
4 main PROC
5 $LN11:
6     mov     QWORD PTR [rsp+16], rdx
7     mov     DWORD PTR [rsp+8], ecx
8     sub     rsp, 56 ; 00000038H
9     mov     QWORD PTR $T1[rsp], rsp
10    lea     rdx, $label$12
11    mov     rcx, QWORD PTR $T1[rsp]
12    call    _local_unwind
13    call    int throw_exception_divide_by_zero(void) ; throw_exception_divide_by_zero()
14    npad    1
15 $LN9@main:
16     lea     rcx, OFFSET FLAT:$SG27156
17     call    printf
18 $label$12:
19     xor     eax, eax
20 $LN5@main:
21     add     rsp, 56 ; 00000038H
22     ret     0
23 main     ENDP

```

Если необходимо просто завершить выполнение блока `__try` без аварийного выхода, то есть не начиная раскрутку стека, то в этом случае нужно использовать инструкцию `__leave`. Программа, использующая `__leave` для выхода из блока `__try`.

```

1  int main()
2  {
3      __try
4      {
5          __leave;
6          throw_exception_divide_by_zero();
7      }
8      __finally
9      {
10         printf("__finally \n");
11     }
12     system("pause");
13     return 0;
14 }

```

Её ассемблерный код приведен ниже. Из этого кода видно, что на месте вызова `__leave`, стоит вызов инструкции `jmp`, которая выполняет раскрутку стека и переводит исполнение программы к маркеру `$LN2@main`.

Листинг 8: Скомпилированный ассемблерный код

```

1  argc$ = 48
2  argv$ = 56
3  main PROC
4  $LN10:
5      mov     QWORD PTR [rsp+16], rdx
6      mov     DWORD PTR [rsp+8], ecx
7      sub     rsp, 40 ; 00000028H
8      jmp     SHORT $LN2@main
9      call    int throw_exception_divide_by_zero(void) ; throw_exception_divide_by_zero
10     npad    1
11 $LN2@main:
12 $LN8@main:
13     lea     rcx, OFFSET FLAT:$SG27154
14     call    printf
15     xor     eax, eax
16 $LN4@main:
17     add     rsp, 40 ; 00000028H
18     ret     0
19 main ENDP

```

### 3.9 Преобразование структурное исключение в исключение языка C

В реализации C++ предусмотрен механизм, который позволяет использовать механизм структурной обработки исключений в механизме обработки исключений, используемом в C++. Для этой цели была разработана функция `_set_se_translator`. Эта функция устанавливает в системе функцию, которая называется функцией-транслятором, назначение которой состоит в том, чтобы преобразовывать структурные исключения в исключения C++. Если функция-транслятор установлена, то она вызывается всегда при выбросе структурного исключения. В функции-трансляторе можно использовать инструкцию `throw`, которая будет выбрасывать исключение нужного типа. Функция-транслятор должна иметь следующий прототип:

```

1  typedef void (*_se_translator_function)(unsigned int, struct _EXCEPTION_POINTERS*);

```

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

```

1  _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_translator` адрес старой функции-транслятора, которая в дальнейшем может быть восстановлена при помощи вызова `_set_se_translator`. Если функция-транслятор устанавливается в первый раз, то возвращаемое значение может быть равно `NULL`. Рассмотрим пример:

```

1  void se_trans_func(unsigned code, EXCEPTION_POINTERS *)
2  {
3      throw code;
4  }

```

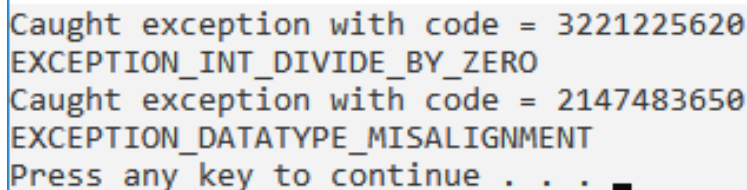
```

1 int main(int argc, char **argv)
2 {
3     _set_se_translator(se_trans_func);
4     try
5     {
6         throw_exception_divide_by_zero();
7     }
8     catch (unsigned code)
9     {
10        handle_exception(code);
11    }
12    try
13    {
14        throw_exception_datatype_misalignment();
15    }
16    catch (unsigned code)
17    {
18        handle_exception(code);
19    }
20    return 0;
21 }

```

Данная программа генерирует заданные исключения. Функция-транслятор просто возвращает нам код возникшего исключения. В зависимости от полученного кода обработчик выводит нам соответствующее сообщение. Следует заметить, что для использования функции `_set_se_translator` необходимо передать компилятору флаг `/EHa`, который разрешает работу с C++ исключениями.

В результате выполнения программы:



```

Caught exception with code = 3221225620
EXCEPTION_INT_DIVIDE_BY_ZERO
Caught exception with code = 2147483650
EXCEPTION_DATATYPE_MISALIGNMENT
Press any key to continue . . . █

```

Рис. 9: Вывод программы

Также в функции-трансляторе можно возвращать более детальную информацию. Например – структуру `EXCEPTION_RECORD`:

```

1 void se_trans_func(unsigned code, EXCEPTION_POINTERS *info)
2 {
3     throw *(info->ExceptionRecord);
4 }

```

```

1 int main()
2 {
3     _set_se_translator(se_trans_func);
4     try
5     {
6         throw_exception_devide_by_zero();
7     }
8     catch (EXCEPTION_RECORD record)
9     {
10        handle_exception(record.ExceptionCode);
11    }
12    try
13    {
14        throw_exception_datatype_misalignment();
15    }
16    catch (EXCEPTION_RECORD record)
17    {
18        handle_exception(record.ExceptionCode);
19    }
20    return 0;
21 }

```

В результате выполнения программы:



```
Caught exception with code = 3221225620
EXCEPTION_INT_DIVIDE_BY_ZERO
Caught exception with code = 2147483650
EXCEPTION_DATATYPE_MISALIGNMENT
Press any key to continue . . . █
```

Рис. 10: Вывод программы

### 3.10 Использование `__try` – `__finally` блока

В Windows существует еще один способ обработки исключений - код, при исполнении которого возможен выбрасывание 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:

```
1  __try
2  {
3      // guarded code
4  }
5  __finally
6  {
7      // finally code
8  }
```

Финализированная 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.

```
1  int main()
2  {
3      __try
4      {
5          __try
6          {
7              __try
8              {
9                  throw_exception_divide_by_zero();
10             }
11             __finally
12             {
13                 std::cout << "First finally" << std::endl;
14             }
15         }
16         __except (filter(GetExceptionCode(), EXCEPTION_INT_DIVIDE_BY_ZERO))
17         {
18             std::cout << "Caught exception: EXCEPTION_INT_DIVIDE_BY_ZERO" << std::endl;
19         }
20         __try
21         {
22             __try
23             {
24                 throw_exception_datatype_misalignment();
25             }
26             __finally
27             {
28                 std::cout << "Second finally" << std::endl;
29             }
30         }
31         __except (filter(GetExceptionCode(), EXCEPTION_DATATYPE_MISALIGNMENT))
32         {
33             std::cout << "Caught exception: EXCEPTION_DATATYPE_MISALIGNMENT" << std::endl;
34         }
35     }
36     __finally
37     {
38         std::cout << "Main Finally" << std::endl;
39     }
40     return 0;
41 }
```

По выводу программы можно понять, что код из блока `__finally` вызывается не только в случае ошибки, но и если блок завершился корректно.

В результате выполнения программы:

```
First finally
Caught exception: EXCEPTION_INT_DIVIDE_BY_ZERO
Second finally
Caught exception: EXCEPTION_DATATYPE_MISALIGNMENT
Main Finally
Press any key to continue . . . █
```

Рис. 11: Вывод программы

Управление из блока `__try` может быть передано одним из следующих способов:

1. нормальное завершение блока
2. выход из блока при помощи управляющей инструкции `__leave`
3. выход из блока при помощи одной из управляющих инструкций `return`
4. `break`, `continue` или `goto C++`
5. передача управления обработчику исключения

В первых двух случаях считается, что блок `__try` завершился нормально, а во вторых двух случаях — аварийно. Для того чтобы определить, как завершился блок `__try`, используется функция `AbnormalTermination`, которая имеет следующий прототип:

```
1  BOOL AbnormalTermination (VOID);
```

В случае если блок `__try` завершился аварийно, эта функция возвращает ненулевое значение, а в противном случае — значение `false`.

```
1  int main(int argc, char **argv) {
2      __try
3      {
4          __try
5          {
6              __try
7              {
8                  throw_exception_divide_by_zero();
9              }
10             __finally
11             {
12                 std::cout << "First finally" << std::endl;
13                 if (AbnormalTermination())
14                 {
15                     std::cout << "Try teminated (first)" << std::endl;
16                 }
17             }
18         }
19         __except (filter(GetExceptionCode(), EXCEPTION_INT_DIVIDE_BY_ZERO))
20         {
21             std::cout << "Caught exception: EXCEPTION_INT_DIVIDE_BY_ZERO" << std::endl;
22         }
23         __try
24         {
25             __try
26             {
27                 throw_exception_datatype_misalignment();
28             }
29             __finally
30             {
31                 std::cout << "Second finally" << std::endl;
32                 if (AbnormalTermination())
33                 {
34                     std::cout << "Try teminated (second)" << std::endl;
35                 }
36             }
37         }
38         __except (filter(GetExceptionCode(), EXCEPTION_DATATYPE_MISALIGNMENT))
39         {
40             std::cout << "Caught exception: EXCEPTION_DATATYPE_MISALIGNMENT" << std::endl;
```

```

41     }
42 }
43 __finally
44 {
45     std::cout << "Main finally" << std::endl;
46     if (AbnormalTermination())
47     {
48         std::cout << "Try terminated (Main)" << std::endl;
49     }
50 }
51 system("pause");
52 return 0;
53 }

```

Из вывода программы видно, что в случае корректного завершения блока `__try`, функция `AbnormalTermination` возвращает `false`.

```

First finally
Try terminated (first)
Caught exception: EXCEPTION_INT_DIVIDE_BY_ZERO
Second finally
Try terminated (second)
Caught exception: EXCEPTION_DATATYPE_MISALIGNMENT
Main finally
Press any key to continue . . . ■

```

Рис. 12: Вывод программы

## 4 Вывод

В ходе работы были изучены структурные исключения SEH. Механизм структурной обработки исключений в Windows немного отличается от механизма обработки исключений, принятого в языке программирования C++. Дело в том, что механизм структурной обработки исключений был разработан раньше, чем принят стандарт языка C++. Кроме того, в отличие от языка программирования C++ 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`. В свою очередь обработчик исключения `catch` может рассматриваться как функция с одним параметром, которая выполняется только в том случае, если тип ее параметра соответствует типу выброшенного исключения.

Из преимуществ данного способа обработки исключений, по сравнению со встроенными средствами языка C++ является:

- возможность обработки аппаратных исключений и просмотра регистров процессора на момент их возникновения;
- поддерживаются как в языке C, так и в C++;
- возможность транслирования исключений в исключения языка C++.

Из минусов стоит отметить:

- зависимость от конкретной платформы, в то время как исключения языка C++ стандартизованы.

**OllyDbg** – бесплатный проприетарный отладчик уровня ассемблера для операционных систем Windows, предназначенный для анализа и модификации откомпилированных исполняемых файлов и библиотек.

OllyDbg выгодно отличается от классических отладчиков (таких, как SoftICE) интуитивно понятным интерфейсом, подсветкой специфических структур кода, простотой в установке и запуске.

**WinDBG** – это многоцелевой отладчик для операционной системы Windows. Его можно использовать для отладки приложений пользовательского режима, драйверов устройств и самой операционной системы в режиме ядра. Как и более известный отладчик Visual Studio, он имеет графический пользовательский интерфейс. Все функциональности, которые представлены в данном инструменте довольно удобные для использования.

**Process Monitor** — утилита, которая имеет возможности программы мониторинга обращений к реестру Regmon и программы мониторинга обращений к файловой системе Filemon, и дополнительно, позволяет получать более подробную информацию о взаимодействии процессов, использовании ресурсов, сетевой активности и операциях ввода-вывода.

Process Monitor выполняет наблюдение в реальном времени для следующих классов событий:

- Файловая система: создание(открытие)/закрытие/чтение/запись/удаление элементов файловой системы: файлов, каталогов, атрибутов, содержимого.
- Реестр: создание/чтение/запись/перечисление/удаление элементов реестра: ветвей, ключей, значений.
- Сеть: установка соединения, передача данных, закрытие соединения. Информация об источнике/-приемнике TCP/UDP трафика. Общая информация о протоколах, пакетах. Сами передаваемые данные не записываются.
- Процесс/поток: Создание процесса, создание потока внутри процесса, завершение потока/процесса. детальная информация о процессе (путь, командная строка, ID пользователя/сессии), запуск/завершение, загрузка образов (библиотеки/драйвера), стек выполнения.
- Профилирование: Специальный класс событий, записываемых с целью отслеживания количества процессорного времени, затрачиваемого каждым процессом. Использование памяти процесса.

Для своей работы, Process Monitor устанавливает в системе собственный драйвер PROCMON20.SYS, с помощью которого выполняется перехват контролируемых монитором системных функций и сбор данных подлежащих мониторингу. Наблюдение выполняется для следующих классов операций - обращения к файловой системе (file system), обращение к реестру (Registry), работа с сетью (Network), и активность процессов (Process).