Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Лабораторная работа №1 (Обработка исключений)**

**Дисциплина**: Системное программирование

Выполнил студент гр. 13541/3 Смирнов М.И.

(подпись)

Руководитель Душутина Е.В.

(подпись)

“ ” 2017 г.

Санкт – Петербург

2017

**Содержание**

1. **Задание**
2. Сгенерировать и обработать исключение с помощь WinAPI функций
3. Получить код исключения с помощью GetExceptionCode двумя способами:
   * использовать эту функцию в выражении фильтре;
   * использовать эту функцию в обработчике;
4. Создать собственную функцию-фильтр.
5. Получить информацию об исключении с помощью GetExceptionInformation. генерировать исключение с помощью RaiseException.
6. Использовать функции InhandleExceptionFilter и SetUnhandleExceptionFilter для необработанных исключений.
7. Обработать вложенные исключения.
8. Выйти из блока \_try с помощью:

* goto;
* leave.

1. Преобразовать структурные исключения (\_try) в исключение языка С. Для этого использовать функцию-транслятор (translator).
2. Использовать финальный обработчик finally.
3. Проверить корректность выхода из блока \_try с помощью функции AbnormalTermination в финальном обработчике finally.
4. **Решение**
5. **Сгенерировать и обработать исключение с помощь WinAPI функций**

Исключение — это событие при выполнении программы, которое приводит к её ненормальному или неправильному поведению. Для того, чтобы выполнение программы продолжилось далее необходимо или аварийное завершение программы, или немедленное её восстановление.

Для операционных системах Windows имеется специальный механизм структурной обработки исключений (structured exception handling). SEH обеспечивает надежность программ, благодаря ему приложения получают возможность реагировать на такие непредсказуемые события, как исключения адресации, арифметические сбои и системные ошибки. Использование SEH позволяет программам осуществлять корректный выход из любой точки программного блока и автоматически выполнять предусмотренную программистом обработку ошибок для восстановления своей работоспособности.

Принцип работы SEH: в программе место, в котором может произойти исключение, выделяется отдельным блоком кода (называют его фреймом).  После фрейма ниже записывают программный блок, который обрабатывает происшедшее исключение (обработчик исключения). По завершению обработки исключения код продолжается далее.

Для этого в C# предназначена конструкция try...catch...finally. Те инструкции программы, где ожидается возможность появления исключительных ситуаций, содержатся в бло­ке try. Если в блоке try возникает исключение, т. е. ошибка, то генерируется исключение. Исклю­чение перехватывается, используя catch, и обрабатывается. Код в блоке finally выполняется даже в том случае, если создано исключение. Далее представлен код 2-х исключений на C#.

Исключение EXCEPTION\_INT\_DIVIDE\_BY\_ZERO (на языке С# DivideByZeroException):

static void Exception\_DivideByZeroException()

{

int result;

int a = 10;

int b = Int16.Parse(Console.ReadLine());

try

{

result = a / b;

Console.WriteLine("{0} / {1} = {2:0.##}", a, b, result);

}

catch (DivideByZeroException e)

{

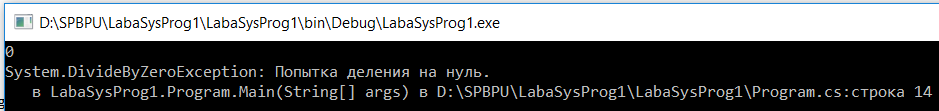
Console.WriteLine(e);

}

Console.ReadKey();

}

Результат при вводе значения “0”:



Исключение IndexOutOfRangeException:

static void Exception\_IndexOutOfRangeException()

{

int[] mass = new int[3] { 0, 1, 2 };

int i = Int16.Parse(Console.ReadLine());

try

{

Console.WriteLine(mass[i]);

}

catch (IndexOutOfRangeException e)

{

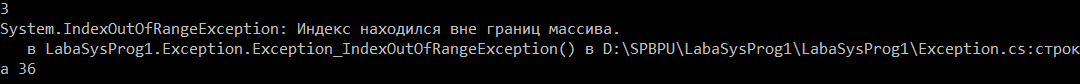
Console.WriteLine(e);

}

Console.ReadKey();

}

Результат при вводе значения “3”:



1. **Получить код исключения с помощью GetExceptionCode двумя способами**
   * 1. использовать эту функцию в обработчике.
     2. использовать эту функцию в выражении фильтре.

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. В языке C# функция GetExceptionCode находиться в классе Marshal.

Назначение функции GetExceptionCode:

* В выражении фильтра она используется для того, чтобы определить выполняет ли текущий обработчик исключения обработку исключений с данным кодом или нужно продолжить поиск подходящего обработчика исключения.
* В блоке обработки исключения она также предназначена для проверки кодов исключений, которые обрабатывает текущий обработчик исключения, но в этом случае поиск другого обработчика исключений не выполняется.

1. Функция GetExceptionCode в обработчике:

static void Exception\_in\_handler()

{

int[] mass = new int[3] { 0, 1, 2 };

int i = Int16.Parse(Console.ReadLine());

int divider=Int16.Parse(Console.ReadLine());

try

{

Console.WriteLine(mass[i]/divider);

}

catch (System.Exception)

{

Console.WriteLine(Marshal.GetExceptionCode());

}

Console.ReadKey();

}

Чтобы использовать Marshal.GetExceptionCode() для вывода номера ошибки необходимо включить библиотеку System.Runtime.InteropServices.

Результат при вводе значения “2” и “0” (попадает под исключение DivideByZeroException):



Результат при вводе значения “3” и “1” (попадает под исключение IndexOutOfRangeException):



1. Функция GetExceptionCode в выражении фильтре:

static void Exception\_in\_filter()

{

int[] mass = new int[3] { 0, 1, 2 };

int i = Int16.Parse(Console.ReadLine());

int divider = Int16.Parse(Console.ReadLine());

try

{

Console.WriteLine(mass[i] / divider);

}

catch (System.Exception) when (divider==0)

{

Console.WriteLine("divider=0");

Console.WriteLine("Error: -1073741676");

}

catch (System.Exception) when (i>2)

{

Console.WriteLine("i>2");

Console.WriteLine("Error: -532462766");

}

catch (System.Exception)

{

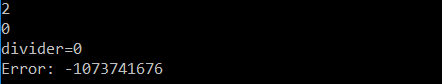
Console.WriteLine(Marshal.GetExceptionCode());

}

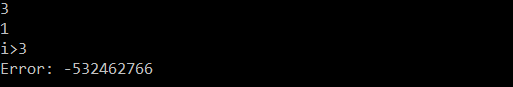
Console.ReadKey();

}

Результат при вводе значения “2” и “0” (попадает под исключение DivideByZeroException):



Результат при вводе значения “3” и “1” (попадает под исключение IndexOutOfRangeException):



Если будет исключение, не вошедшее в условия divider=0 и i>3, программа зайдет в последний catch и выдаст просто номер ошибки.

1. **Создать собственную функцию-фильтр**

Можно использовать так называемую функцию-фильтр, если для принятия решения об обработке исключения требуется более детально обработать информацию об исключении. Функции GetExceptionCode и GetExceptionlnformation в функции фильтра вызывать не разрешается, хотя эти функции могут вызываться для инициализации параметров функции фильтра. Но это работает только на C и C++. В C# можно это заменить использованием комбинации if и throw (Оператор throw используется для сообщения о случаях аномальных ситуаций (исключений) в ходе выполнения программы).

Далее рассмотрим такой пример в C# для исключений DivideByZeroException и IndexOutOfRangeException:

static void Main(string[] args)

{

int[] mass = new int[3] { 0, 1, 2 };

int i = Int16.Parse(Console.ReadLine());

int divider = Int16.Parse(Console.ReadLine());

try

{

Console.WriteLine(Metod(mass,divider,i));

}

catch (DivideByZeroException)

{

Console.WriteLine("Error! divider=0");

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("Error! i>2");

}

Console.ReadKey();

}

static public int Metod (int[] mass, int divider, int i)

{

if (divider==0)

throw new DivideByZeroException();

if (i > 2)

throw new IndexOutOfRangeException();

return mass[i] / divider;

}

1. **Получить информацию об исключении с помощью GetExceptionInformation. Генерировать исключение с помощью RaiseException.**

Так как язык С# более высокоуровневый (нежели С++ и С), у него нет таких функций, какGetExceptionInformation и RaiseException. Поэтому далее мы перейдем на язык C++.

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

LPEXCEPTION\_POINTERS GetExceptionlnformation(void);

Возвращаемая структура содержит два указателя на структуры: Context и Exception\_Record.

1. 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.
2. В структуру Exception\_Record система записывает информацию об исключении. Структура имеет следующий формат:

typedef struct EXCEPTION\_RECORD {

DWORD ExceptionCode;

DWORD ExceptionFlags,

strict \_EXCEPTION\_RECORD \*ExceptionRecord;

PVOID ExceptionAddress,

DWORD Number Parameters,

ULONG\_PTR Exceptionlnformation [ EXCEPTION\_MAXIMUM\_PARAMETERS ] ;

} EXCEPTION\_RECORD, \*PEXCEPTION\_RECORD;

Поля этой структуры имеют следующее назначение:

* ExceptionCode содержит код исключения, который может принимать такие же значения, как и код исключения, возвращаемый функцией GetExceptionCode.
* ExceptionFlags может принимать одно из двух значений:
* 0 — обозначает, что после обработки исключения возможно возобновление выполнения программы;
* EXCEPTION\_NONCONTINUABLE — обозначает, что после обработки исключения возобновление выполнения программы невозможно. Если это значение установлено и выполнена попытка возобновления выполнения программы, то система выдаст исключение EXCEPTION\_NONCONTINUABLE\_EXCEPTION.
* ExceptionRecord содержит указатель на следующую структуру типа exception\_record, которая может быть создана в случае вложенных исключений.
* ExceptionAddress содержит адрес инструкции в программе, на которой произошло исключение.
* NumberParameters содержит количество параметров, заданных в поле Exceptioninformation, которое является последним в этой структуре.

Поле Exceptionlnformation[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 GetExceptionlnformation 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 GetExceptionlnformation можно использовать для двух целей:

1. Получение более подробной информации об исключении, учитывая содержимое структуры типа context;
2. Обработка вложенных исключений.

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

Исключение EXCEPTION\_INT\_DIVIDE\_BY\_ZERO

#include <windows.h>

#include <iostream>

using namespace std;

EXCEPTION\_RECORD er; // информация об исключении

DWORD filter\_function(EXCEPTION\_POINTERS \*p)

{

// сохраняем содержимое структуры EXCEPTION\_RECORD

er = \*(p->ExceptionRecord);

return EXCEPTION\_EXECUTE\_HANDLER;

}

int main()

{

\_try

{

RaiseException(

3221225620, // код исключения деления на ноль

0, // continuable exception

0, NULL); // no arguments

}

\_except(filter\_function(GetExceptionInformation()))

{

// распечатываем информацию об исключении

cout << "ExceptionCode = " << er.ExceptionCode << endl;

cout << "ExceptionFlags = " << er.ExceptionFlags << endl;

cout << "ExceptionRecord = " << er.ExceptionRecord << endl;

cout << "ExceptionAddress = " << er.ExceptionAddress << endl;

cout << "NumberParameters = " << er.NumberParameters << endl;

// распечатываем параметры

if (er.ExceptionCode == EXCEPTION\_INT\_DIVIDE\_BY\_ZERO)

{

cout << "Type of access = " << er.ExceptionInformation[0] << endl;

cout << "Address of access = " << er.ExceptionInformation[1] << endl;

}

cout << endl;

}

return 0;

}

В результате выполнения, получили следующий результат:

ExceptionCode = 3221225620

ExceptionFlags = 0

ExceptionRecord = 00000000

ExceptionAddress = 00F15F48

NumberParameters = 0

Type of access = 0

Address of access = 0

### Исключение EXCEPTION\_FLT\_OVERFLOW

Листинг 2.8 Исключение EXCEPTION\_FLT\_OVERFLOW

#include <windows.h>

#include <iostream>

using namespace std;

EXCEPTION\_RECORD er; // информация об исключении

DWORD filter\_function(EXCEPTION\_POINTERS \*p)

{

// сохраняем содержимое структуры EXCEPTION\_RECORD

er = \*(p->ExceptionRecord);

return EXCEPTION\_EXECUTE\_HANDLER;

}

int main()

{

\_try

{

RaiseException( // вызываем исключение

3221225617, // код исключения деления на ноль

0, // continuable exception

0, NULL); // no arguments

}

\_except(filter\_function(GetExceptionInformation()))

{

// распечатываем информацию об исключении

cout << "ExceptionCode = " << er.ExceptionCode << endl;

cout << "ExceptionFlags = " << er.ExceptionFlags << endl;

cout << "ExceptionRecord = " << er.ExceptionRecord << endl;

cout << "ExceptionAddress = " << er.ExceptionAddress << endl;

cout << "NumberParameters = " << er.NumberParameters << endl;

// распечатываем параметры

if (er.ExceptionCode == EXCEPTION\_INT\_DIVIDE\_BY\_ZERO)

{

cout << "Type of access = " << er.ExceptionInformation[0] << endl;

cout << "Address of access = " << er.ExceptionInformation[1] << endl;

}

cout << endl;

}

return 0;

}

В результате выполнения, получили следующий результат

ExceptionCode = 3221225617

ExceptionFlags = 0

ExceptionRecord = 00000000

ExceptionAddress = 77AF2EEC

NumberParameters = 0

**Список литературы**

1. Интуит. Национальный открытый университет [Электронный ресурс]. – Режим доступа: http://www.intuit.ru/studies/courses/46/46/lecture/1368 (Дата обращение: 22.09.17).
2. Психологос. Энциклопедия практической психологии [Электронный ресурс]. – Режим доступа: http://www.psychologos.ru/articles/view/razum (Дата обращение: 22.09.17).
3. Энциклопедия МедПортал [Электронный ресурс]. – Режим доступа: http://medportal.ru/enc/psychology/personalitypsychology/13/ (Дата обращения: 22.09.17).
4. Отделение интеллектуальных систем в гуманитарной сфере РГГУ [Электронный ресурс]. – Режим доступа: http://ois.rsuh.ru/Special\_IS.htm (Дата обращение: 22.09.17).