Exception handling features help you deal with any unexpected or **exceptional situations** that occur when a program is running

HTTPS://DOCS.MICROSOFT.COM/EN-US/DOTNET/CSHARP/PROGRAMMING-GUIDE/EXCEPTIONS/

Internals of Exceptions

CONTACT@ADAMFURMANEK.PL

HTTP://BLOG.ADAMFURMANEK.PL

FURMANEKADAM

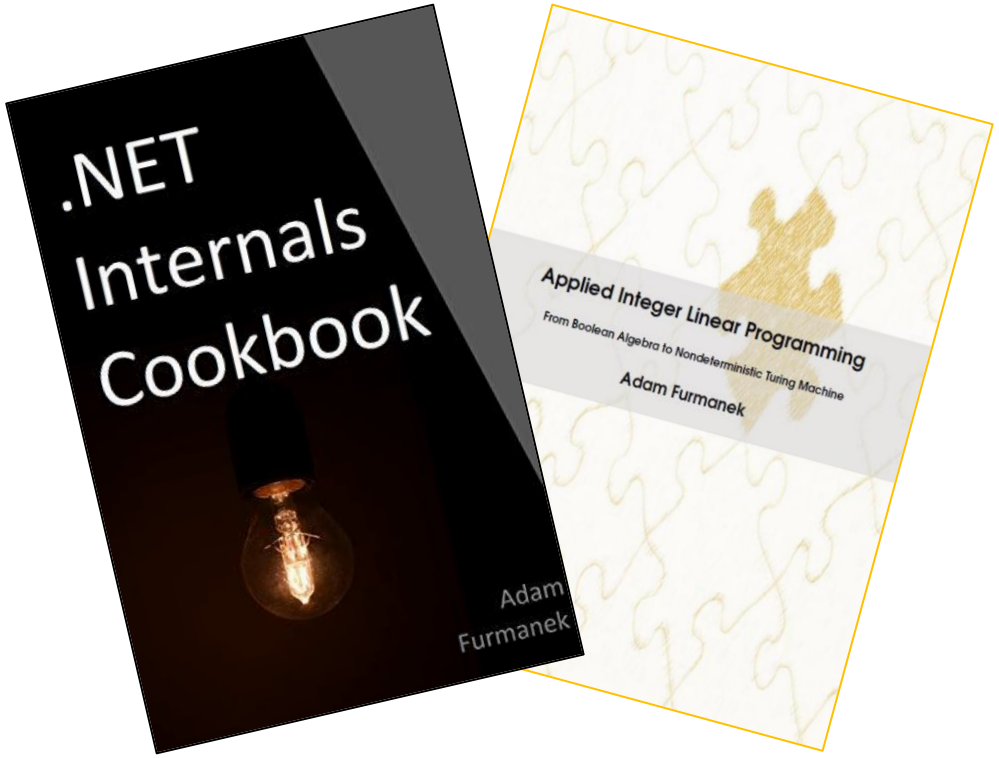
About me

Software Engineer, Blogger, Book Writer, Public Speaker.
Author of ***Applied Integer Linear Programming*** and ***.NET Internals Cookbook***.

http://blog.adamfurmanek.pl

contact@adamfurmanek.pl

furmanekadam



Random IT Utensils

IT, operating systems, maths, and more.

Agenda

Exception mechanisms overview

Implementation details:

- How to rethrow, catch everything, and what cannot be handled?
- When are exceptions thrown and how to stop it?
- How to handle corrupted state?

Even more implemenation details:

- SEH and its frame.
- .NET and double pass.
- Thread.Abort implementation.
- AccessViolation internals.
- VEH and machine code generation to catch StackOverflowException

Exception mechanisms overview

Popular mechanisms

C++

```
try {
    ...
} catch (std::exception& e) {
    ...
}
```

C#

```
try {
    ...
} catch (Exception e)
  when (e.Message == "Custom") {
    ...
} finally {
    ...
}
```

IL handlers

try, finally — the same as in C#.

Catch:

- Inheritance order is not enforced by CLR, it is checked by the C# compiler only.

Fault:

- Executed always when there was an exception (similar to finally).

Filter:

- Indicates whether you want to handle the exception or not.
- This can be just a typecheck of some other logic (like content check).

```
.try { ... }
catch [mscorlib]System.Exception { ... }
finally { ... }
fault { ... }
filter { conditions } { code }
```

Lower level mechanisms

WINAPI, VSC++, GCC

Structured Exception Handling (SEH).

Vectored Exception Handling (VEH), Vectored Continuation Handling (VCH).

DWARF:

- Similar to SEH.
- Table based.

SetJump LongJump:

- Restores registers.

SEH

```
__try {
    ...
} __except (filter) {
    ...
}


__try {
    ...
} __finally {
    ...
}
```

```c
#include "stdafx.h"
#include <Windows.h>

void GenerateException(){
    int divideByZero = 0;
    printf("\tDivided! %d", 1 / divideByZero);
}

int FilterException(){
    static int count = 0;
    if(count ++ < 10){
        printf("\tContinuing for the %d time\n", count);
        return EXCEPTION_CONTINUE_EXECUTION;
    };

    return EXCEPTION_EXECUTE_HANDLER;
}

void Try(){
    __try{
        puts("Entering try");
        GenerateException();
        puts("Exiting try");
    }
    __except(FilterException()){
        puts("In __except");
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    Try();
    return 0;
}
```

C:\WINDOWS\system32\cmd.exe

```
Entering try
        Continuing for the 1 time
        Continuing for the 2 time
        Continuing for the 3 time
        Continuing for the 4 time
        Continuing for the 5 time
        Continuing for the 6 time
        Continuing for the 7 time
        Continuing for the 8 time
        Continuing for the 9 time
        Continuing for the 10 time
In __except
Press any key to continue . . .
```

```
1  #include "stdafx.h"
2  #include <windows.h>
3
4  int divideByZero = 0;
5
6  LONG WINAPI VehExceptionHandler(_EXCEPTION_POINTERS* pExceptionInfo){
7      puts("In VEH exception handler");
8
9      return EXCEPTION_CONTINUE_SEARCH;
10
11     //divideByZero = 1;
12     //return EXCEPTION_CONTINUE_EXECUTION;
13 }
14
15 LONG WINAPI VchContinuationHandler(_EXCEPTION_POINTERS* pExceptionInfo) {
16     puts("IN VCH continuation handler");
17
18     return EXCEPTION_CONTINUE_EXECUTION;
19 }
20
21 void GenerateException(){
22     divideByZero = 1 / divideByZero;
23 }
24
25 void Try(){
26     __try{
27         puts("Entering try");
28         GenerateException();
29         puts("Exiting try");
30     }
31     __except(EXCEPTION_EXECUTE_HANDLER){
32         puts("In __except");
33     }
34 }
35
36 int _tmain(int argc, _TCHAR* argv[])
37 {
38     // __except called when VEH doesn't handle Exception
39     // VCH called when VEH handles the exception
40
41     AddVectoredExceptionHandler(1, VehExceptionHandler);
42     AddVectoredContinueHandler(1, VchContinuationHandler);
43
44     Try();
45
46     return 0;
47 }
48
```

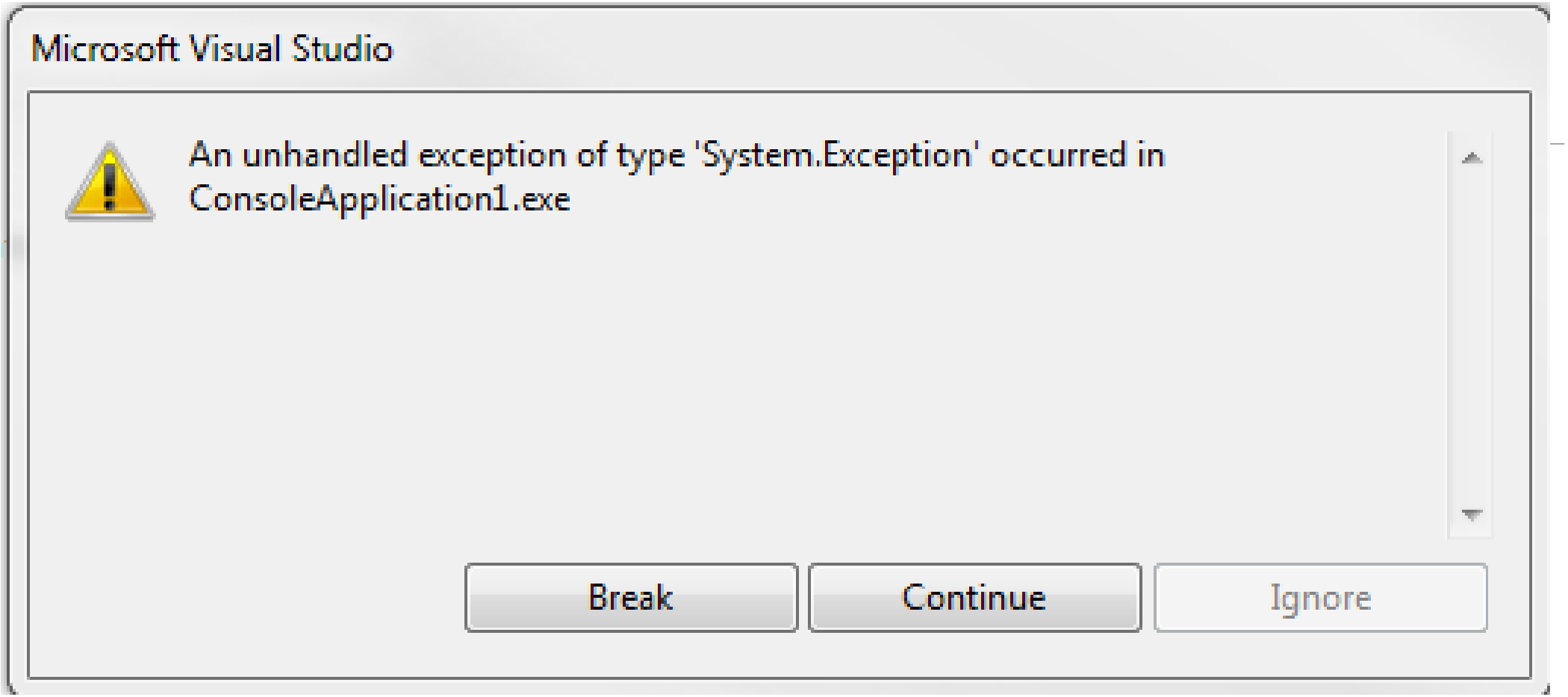
C:\WINDOWS\system32\cmd.exe

```
Entering try
In VEH exception handler
In __except
Press any key to continue . . .
```

```
#include "stdafx.h"
#include <windows.h>

int divideByZero = 0;

LONG WINAPI VehExceptionHandler(_EXCEPTION_POINTERS* pExceptionInfo){
    puts("In VEH exception handler");

    //return EXCEPTION_CONTINUE_SEARCH;

    divideByZero = 1;
    return EXCEPTION_CONTINUE_EXECUTION;
}

LONG WINAPI VchContinuationHandler(_EXCEPTION_POINTERS* pExceptionInfo) {
    puts("IN VCH continuation handler");

    return EXCEPTION_CONTINUE_EXECUTION;
}

void GenerateException(){
    divideByZero = 1 / divideByZero;
}

void Try(){
    __try{
        puts("Entering try");
        GenerateException();
        puts("Exiting try");
    }
    __except(EXCEPTION_EXECUTE_HANDLER){
        puts("In __except");
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    // __except called when VEH doesn't handle Exception
    // VCH called when VEH handles the exception

    AddVectoredExceptionHandler(1, VehExceptionHandler);
    AddVectoredContinueHandler(1, VchContinuationHandler);

    Try();

    return 0;
}
```

C:\WINDOWS\system32\cmd.exe

```
Entering try
In VEH exception handler
IN VCH continuation handler
Exiting try
Press any key to continue . . .
```


Microsoft Visual Studio
An unhandled exception of type 'System.Exception' occurred in ConsoleApplication1.exe
Break
Continue
Ignore

One might ask…

1. What value is returned if I return in both try and finally? What happens if I throw in those places?
2. Is finally executed if I have an unhandled exception?
3. Is finally executed if I exit the application by calling the *exit* or *Environment.Exit*?
4. If it is how can I disable it? And what if I want only some of the finally blocks to be executed?
5. Is finally executed if I have an *OutOfMemoryException*?
6. Can I catch an *AccessViolation*? *ThreadAbort*? *StackOverflow*? *ExecutionEngine*?
7. Can I throw a non-exception in C#? Can I catch a non-exception in C#?
8. What happens if I throw an exception in a catch block?
9. How do I catch an exceptions from the other threads? What about *async*? What about thread pools?
10. Can I catch an exception during *P/Invoke*?
11. Can I call an async method in *catch* or *finally*?

Implementation details

Stacktraces in .NET Framework

```
07. void ThrowException()
08. {
09.   throw new InvalidOperationException("Invalid");
10. }
11.
12. void RethrowException()
13. {
14.   try
15.   {
16.     ThrowException();
17.     ThrowException();
18.   }
19.   catch (InvalidOperationException e)
20.   {
21.     if (e.Message != "Message")
22.     {
23.       ...
24.     }
25.   }
26. }
27.
28. void Main()
29. {
30.   RethrowException();
31. }
```

```
07. void ThrowException()
08. {
09.   throw new InvalidOperationException("Invalid");
10. }
11.
12. void RethrowException()
13. {
14.   try
15.   {
16.     ThrowException();
17.     ThrowException();
18.   }
19.   catch (InvalidOperationException e)
20.   {
21.     if (e.Message != "Message")
22.     {
23.       throw e;
24.     }
25.   }
26. }
27.
28. void Main()
29. {
30.   MethodRethrowingException();
31. }
```

```
Unhandled Exception: System.InvalidOperationException: Invalid
 at RethrowException () in Program.cs:line 23
 at Main() in Program.cs:line 30
```

```
0:000> !clrstack
00f3ee4c 755bdae8 [HelperMethodFrame: 00f3ee4c]
00f3eefc  01580537 RethrowException() [Program.cs @ 25]
00f3eff0   0158049c Main() [Program.cs @ 30]
```

```
07. void ThrowException()
08. {
09.   throw new InvalidOperationException("Invalid");
10. }
11.
12. void RethrowException()
13. {
14.   try
15.   {
16.     ThrowException();
17.     ThrowException();
18.   }
19.   catch (InvalidOperationException e)
20.   {
21.     if (e.Message != "Message")
22.     {
23.       throw;
24.     }
25.   }
26. }
27.
28. void Main()
29. {
30.   MethodRethrowingException();
31. }
```

```
Unhandled Exception: System.InvalidOperationException: Invalid
 at ThrowException () in Program.cs:line 9
 at RethrowException () in Program.cs:line 23
 at Main() in Program.cs:line 30
```

```
0:000> !clrstack
00f3ee4c 755bdae8 [HelperMethodFrame: 00f3ee4c]
00f3eefc  01580537 RethrowException() [Program.cs @ 25]
00f3eff0   0158049c Main() [Program.cs @ 30]
```

```
07. void ThrowException()
08. {
09.   throw new InvalidOperationException("Invalid");
10. }
11.
12. void RethrowException()
13. {
14.   try
15.   {
16.     ThrowException();
17.     ThrowException();
18.   }
19.   catch (InvalidOperationException e)
20.   {
21.     if (e.Message != "Message")
22.     {
23.       throw new Exception(e.Message, e);
24.     }
25.   }
26. }
27.
28. void Main()
29. {
30.   MethodRethrowingException();
31. }
```

```
Unhandled Exception: System.Exception: Invalid
  at ThrowException () in Program.cs:line 9
  at RethrowException () in Program.cs:line 16
  --- End of inner exception stack trace ---
  at RethrowException () in Program.cs:line 23
  at Main() in Program.cs:line 30
```

```
0:000> !clrstack
00f3ee4c 755bdae8 [HelperMethodFrame: 00f3ee4c]
00f3eefc  01580537 RethrowException() [Program.cs @ 25]
00f3eff0   0158049c Main() [Program.cs @ 30]
```

```
07. void ThrowException()
08. {
09.   throw new InvalidOperationException("Invalid");
10. }
11.
12. void RethrowException()
13. {
14.   try
15.   {
16.     ThrowException();
17.     ThrowException();
18.   }
19.   catch (InvalidOperationException e)
20.   {
21.     if (e.Message != "Message")
22.     {
23.       typeof(Exception).GetMethod("PrepForRemoting", BindingFlags.NonPublic | BindingFlags.Instance).Invoke(new object[0]);
24.       throw e;
25.     }
26.   }
27. }
28.
29. void Main()
30. {
31.   MethodRethrowingException();
32. }
```

```
Unhandled Exception: System.InvalidOperationException: Invalid
Server stack trace:
  at ThrowException () in Program.cs:line 9
  at RethrowException () in Program.cs:line 16

Exception rethrown at [0]:
  at RethrowException () in Program.cs:line 24
  at Main() in Program.cs:line 31
```

```
0:000> !clrstack
00f3ee4c 755bdae8 [HelperMethodFrame: 00f3ee4c]
00f3eefc  01580537 RethrowException() [Program.cs @ 26]
00f3eff0   0158049c Main() [Program.cs @ 31]
```

```
07. void ThrowException()
08. {
09.   throw new InvalidOperationException("Invalid");
10. }
11.
12. void RethrowException()
13. {
14.   try
15.   {
16.     ThrowException();
17.     ThrowException();
18.   }
19.   catch (InvalidOperationException e)
20.   {
21.     if (e.Message != "Message")
22.     {
23.       ExceptionDispatchInfo.Capture(e).Throw();
24.     }
25.   }
26. }
27.
28. void Main()
29. {
30.   MethodRethrowingException();
31. }
```

```
Unhandled Exception: System.InvalidOperationException: Invalid
Server stack trace:
  at ThrowException () in Program.cs:line 9
  at RethrowException () in Program.cs:line 16
--- End of stack trace from previous location where exception was thrown ---
  at RethrowException () in Program.cs:line 23
  at Main() in Program.cs:line 30
```

```
0:000> !clrstack
00f3ee4c 755bdae8 [HelperMethodFrame: 00f3ee4c]
00f3eefc 01580537 RethrowException() [Program.cs @ 23]
00f3eff0   0158049c Main() [Program.cs @ 30]
```

Stacktraces in .NET Core

```
07. void ThrowException()
08. {
09.   throw new InvalidOperationException("Invalid");
10. }
11.
12. void RethrowException()
13. {
14.   try
15.   {
16.     ThrowException();
17.     ThrowException();
18.   }
19.   catch (InvalidOperationException e)
20.   {
21.     if (e.Message != "Message")
22.     {
23.       ...
24.     }
25.   }
26. }
27.
28. void Main()
29. {
30.   RethrowException();
31. }
```

```
07. void ThrowException()
08. {
09.   throw new InvalidOperationException("Invalid");
10. }
11.
12. void RethrowException()
13. {
14.   try
15.   {
16.     ThrowException();
17.     ThrowException();
18.   }
19.   catch (InvalidOperationException e)
20.   {
21.     if (e.Message != "Message")
22.     {
23.       throw e;
24.     }
25.   }
26. }
27.
28. void Main()
29. {
30.   MethodRethrowingException();
31. }
```

```
Unhandled Exception: System.InvalidOperationException: Invalid
 at RethrowException () in Program.cs:line 25
 at Main() in Program.cs:line 30
```

```
0:000> !clrstack
00f3eefc_01580537 RehrowException() [Program.cs @ 25]
00f3ee4c 755bdae8 [HelperMethodFrame: 00f3ee4c]
00f3eefc_01580537 ThrowException() [Program.cs @ 8]
00f3eefc  01580537 RethrowException() [Program.cs @ 16]
00f3eff0  0158049c Main() [Program.cs @ 30]
```

```
07. void ThrowException()
08. {
09.   throw new InvalidOperationException("Invalid");
10. }
11.
12. void RethrowException()
13. {
14.   try
15.   {
16.     ThrowException();
17.     ThrowException();
18.   }
19.   catch (InvalidOperationException e)
20.   {
21.     if (e.Message != "Message")
22.     {
23.       throw;
24.     }
25.   }
26. }
27.
28. void Main()
29. {
30.   MethodRethrowingException();
31. }
```

```
Unhandled Exception: System.InvalidOperationException: Invalid
  at ThrowException () in Program.cs:line 9
  at RethrowException () in Program.cs:line 16
  at Main() in Program.cs:line 30
```

```
0:000> !clrstack
00f3eefc_01580537 RehrowException() [Program.cs @ 25]
00f3ee4c 755bdae8 [HelperMethodFrame: 00f3ee4c]
00f3eefc_01580537 ThrowException() [Program.cs @ 8]
00f3eefc  01580537 RethrowException() [Program.cs @ 16]
00f3eff0   0158049c Main() [Program.cs @ 30]
```

Catching everything

C++ — use ellipsis (...).

SEH, VEH — it just works.

.NET — catching *Exception* may not be enough:

- In .NET 2 things thrown in native code are wrapped with *RuntimeWrappedException*.
- In .NET 1 you had to catch them without specifying the type, so catch without the type and catch catching *Exception* were different.
- The code doesn't compile anymore, use *RuntimeCompatibilityAttribute*. It will still be wrapped but then automatically unwrapped.

Java — catching *Throwable* works but is dangerous.

Uncatchable exceptions in .NET

StackOverflowException:

- Could be caught in .NET 1 if occured in managed code.
- Cannot be handled in .NET 2 by default.
- *HandleProcessCorruptedStateExceptionsAttribute* has no effect.
- When loading CLR (i.e., via shim) you can specify if SOE unloads the app domain.
- Use *TryEnsureSufficientExecutionStack* method.

ThreadAbortException:

- Can be caught but not swallowed (it is reraised).
- Can be stopped using *ResetAbort()*.
- Cannot happen in finally block.

AccessViolationException:

- Requires special care (more info later).

ExecutionEngineException:

- You have bigger challenges.

SEHException.

OutOfMemoryException:

- If thrown by the CLR.

Using

```
public class Program
{
    public static void Main()
    {
        try{
            using(var resource = new Resource()){
                Console.WriteLine("Using");
                throw new Exception("Using failed");
            }
        }catch(Exception e){
            Console.WriteLine("Exception: " + e);
        }
    }
}

public class Resource : IDisposable {
    public void Dispose(){
        Console.WriteLine("Disposing");
        throw new Exception("Disposing failed");
    }
}
```

```
Using
Disposing
Exception: System.Exception: Disposing failed
   at Resource.Dispose()
   at Program.Main()
```

Try with resources

```java
import java.util.*;
import java.lang.*;
import java.io.*;

class Ideone
{
    public static void main (String[] args) throws java.lang.Exception
    {
        try(Resource resource = new Resource()){
            System.out.println("Trying");
            throw new RuntimeException("Trying failed");
        }
    }
}

class Resource implements Closeable {
    public void close(){
        System.out.println("Closing");
        throw new RuntimeException("Closing failed");
    }
}
```

```
Trying
Closing
Exception in thread "main" java.lang.RuntimeException: Trying failed
    at Ideone.main(Main.java:11)
    Suppressed: java.lang.RuntimeException: Closing failed
        at Resource.close(Main.java:19)
        at Ideone.main(Main.java:9)
```

Fixing Using

```
public class Program
{
    public static void Main()
    {
        Exception exception = null;
        Exception exception2 = null;
        var resource = new Resource();
        try{
            Console.WriteLine("Using");
            throw new Exception("Using failed");
        }catch(Exception e){
            exception = e;
        }finally{
            try{
                if(resource != null) resource.Dispose();
            } catch(Exception e2){
                exception2 = e2;
            }

            if(exception != null && exception2 != null){
                throw new AggregateException(exception, exception2);
            }else if(exception != null){
                ExceptionDispatchInfo.Capture(exception).Throw();
            }else if(exception2 != null){
                ExceptionDispatchInfo.Capture(exception2).Throw();
            }
        }
    }
}
```

```
Using
Disposing
Unhandled exception. System.AggregateException: One or more errors occurred. (Using failed) (Disposing failed)
 ---> System.Exception: Using failed
   at Program.Main()
   --- End of inner exception stack trace ---
   at Program.Main()
 ---> (Inner Exception #1) System.Exception: Disposing failed
   at Resource.Dispose()
   at Program.Main()<---
```

```
Using
Disposing
Unhandled exception. System.Exception: Disposing failed
   at Resource.Dispose()
   at Program.Main()
--- End of stack trace from previous location where exception was thrown ---
   at Program.Main()
```

```
Using
Disposing
Unhandled exception. System.Exception: Using failed
   at Program.Main()
--- End of stack trace from previous location where exception was thrown ---
   at Program.Main()
```

Fixing Using with ThreadAbort

```
using System;
using System.Threading;
using System.Runtime.ExceptionServices;

public class FixingUsing
{
    public static void Main()
    {
        var handle1 = new ManualResetEvent(false);
        var handle2 = new ManualResetEvent(false);
        var abortedThread = new Thread(() => {
            try
            {
                Exception exception = null;
                Exception exception2 = null;
                var resource = new Resource();
                try
                {
                    Console.WriteLine("Using");
                    handle1.Set();
                    handle2.WaitOne(); // Interruption comes here
                }
                catch (Exception e)
                {
                    exception = e;
                }
                finally
                {
                    try
                    {
                        if (resource != null) resource.Dispose();
                    }
                    catch (Exception e2)
                    {
                        exception2 = e2;
                    }

                    var isAborted = Thread.CurrentThread.ThreadState == ThreadState.AbortRequested;
                    Action<object> killMe = (object o) => {
                        Thread.ResetAbort();
                        if (Type.GetType("Mono.Runtime") != null)
                        {
                            var threadToKill = Thread.CurrentThread;
                            var killer = new Thread(() =>
                            {
                                threadToKill.Abort(o);
                            });
                            killer.Start();
                            killer.Join();
                        }
                        else
                        {
                            Thread.CurrentThread.Abort(o);
                        }
                    };
                    if (exception != null && exception2 != null)
                    {
                        var aggregate = new AggregateException(exception, exception2);
                        if (isAborted)
                        {
                            killMe(aggregate);
                        }
                        else
                        {
                            throw aggregate;
                        }
                    }
                    else if (exception != null)
                    {
                        if (isAborted)
                        {
                            killMe(exception);
                        }
                        else
                        {
                            ExceptionDispatchInfo.Capture(exception).Throw();
                        }
                    }
                    else if (exception2 != null)
                    {
                        if (isAborted)
                        {
                            killMe(exception);
                        }
                        else
                        {
                            ExceptionDispatchInfo.Capture(exception2).Throw();
                        }
                    }
                }
            }
            catch (ThreadAbortException e3)
            {
                Console.WriteLine("Swallowing everything! " + e3.ExceptionState);
            }

            Console.WriteLine("This shouldn't appear");
        });

        abortedThread.Start();
        handle1.WaitOne();
        abortedThread.Abort("With old data");
        handle2.Set();
        abortedThread.Join();
    }
}

public class Resource : IDisposable
{
    public void Dispose()
    {
        Console.WriteLine("Disposing");
        throw new Exception("Disposing failed");
    }
}
```

Output:

```
Using
Disposing
Swallowing everything! System.AggregateException: One or more errors occurred. (Thread was being aborted.) (Disposing failed) ---> System.Thread
   at Program+<>c__DisplayClass0_0.<Main>b__0 () [0x00033] in <0bf37bf98ae6495e939833a76ab993c0>:0
   at System.ExceptionDispatchInfo.Throw() [0x00000] in <0bf37bf98ae6495e939833a76ab993c0>:0 <---
---> (Inner Exception #1) System.ThreadAbortException: Thread was being aborted.
   at Program+<>c__DisplayClass0_0.<Main>b__0 () [0x00033] in <0bf37bf98ae6495e939833a76ab993c0>:0 <---

---> (Inner Exception #2) System.Exception: Disposing failed
   at Resource.Dispose () [0x0000c] in <0bf37bf98ae6495e939833a76ab993c0>:0
   at Program+<>c__DisplayClass0_0.<Main>b__0 () [0x00048] in <0bf37bf98ae6495e939833a76ab993c0>:0 <---
```

```
try
{
    Exception firstException = null;

    var resource = new Resource();
    try
    {
        Console.WriteLine("Using");
        handle1.Set();
        handle2.WaitOne(); // Interruption comes here
    }
    catch (Exception e) when (ExceptionFilter(e, ref firstException))
    {
    }
    finally
    {
        try
        {
            if (resource != null) resource.Dispose();
        }
        catch (Exception e2) when(ExceptionModifier(e2, firstException))
        {
        }
    }
}
catch (ThreadAbortException e3)
{
    Console.WriteLine("Swallowing everything! " + e3.ExceptionState);
}

Console.WriteLine("This shouldn't appear");
```

C:\windows\system32\cmd.exe

```
Using
Disposing

Unhandled Exception: System.Exception: Disposing failed ---> System.Threading.ThreadAbortException: Thread was being abo
rted.
   at System.Threading.WaitHandle.WaitOneNative(SafeHandle waitableSafeHandle, UInt32 millisecondsTimeout, Boolean hasTh
readAffinity, Boolean exitContext)
   at System.Threading.WaitHandle.InternalWaitOne(SafeHandle waitableSafeHandle, Int64 millisecondsTimeout, Boolean hasT
hreadAffinity, Boolean exitContext)
   at System.Threading.WaitHandle.WaitOne(Int32 millisecondsTimeout, Boolean exitContext)
   at System.Threading.WaitHandle.WaitOne()
   at FixingUsing2.<>c__DisplayClass0_0.<Main>b__0() in C:\Users\afish\Desktop\msp_windowsinternals\FixingUsing2\Program
.cs:line 20
   --- End of inner exception stack trace ---
   at Resource.Dispose() in C:\Users\afish\Desktop\msp_windowsinternals\FixingUsing2\Program.cs:line 74
   at FixingUsing2.<>c__DisplayClass0_0.<Main>b__0() in C:\Users\afish\Desktop\msp_windowsinternals\FixingUsing2\Program
.cs:line 29
   at System.Threading.ThreadHelper.ThreadStart_Context(Object state)
   at System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object
state, Boolean preserveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state, B
oolean preserveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state)
   at System.Threading.ThreadHelper.ThreadStart()
Press any key to continue . . .
```

```
1 reference
private static bool ExceptionFilter(Exception e, ref Exception target)
{
    target = e;
    return false;
}

1 reference
private static bool ExceptionModifier(Exception e, Exception first)
{
    if(first != null)
    {
        var newInner = e.InnerException != null ? new AggregateException(first, e.InnerException) : first;
        typeof(Exception).GetField("_innerException", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance).SetValue(e, newInner);
    }

    return false;
}
```

Finally during exit

1. C#
   1. Finally **is not** executed after *Environment.Exit*() and is terminated if was executing.
   2. Finally **is not** executed after *Environment.FailFast()* or rude termination (i.e., via WinAPI).
   3. Finally **is not** executed on *StackOverflowException*, you cannot catch it, it kills the application.
2. SEH
   1. Finally **is not** executed on *exit()*.
3. Java
   1. Finally **is not** executed if *System.exit()* succeeds.
   2. If you have *System.exit* in catch/finally, next statements are not executed.
   3. There is a method *Runtime.runFinalizersOnExit()* but is deprecated and dangerous.
   4. Finally is executed on *StackOverflowException*, you can actually catch it!

Finalizer during exit

It may be executed when exiting with *Environment.Exit()*.

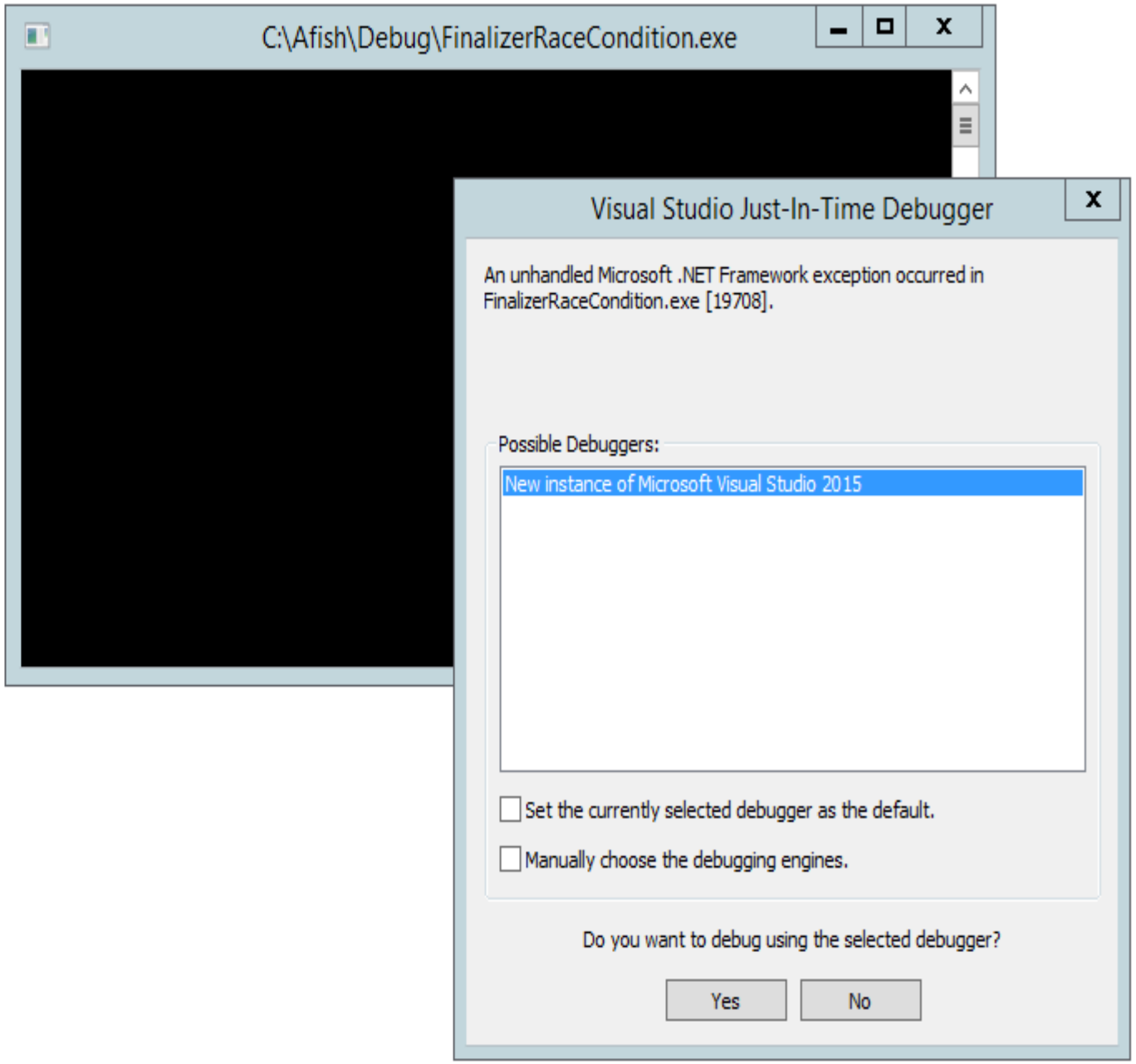
It is not executed with *Environment.FailFast()*.

Finalizing thread cannot run indefinitely. It will be terminated if it takes too long.

It is not reliable. It works differently between .NET Framework and .NET Core for the same code.

Finalizer may be called when the variable is still being used.

```
{
    var content = @"abc";

    if (args.Length > 0)
    {
        TestFile(content);
    }
    else
    {
        while (true)
        {
            Process.Start("FinalizerRaceCondition.exe", "slave").WaitForExit();
            if (new FileInfo("0").Length - 3 > 0)
            {
                Debugger.Launch();
                Debugger.Break();
                Thread.Sleep(100000000);
            }
            Thread.Sleep(1);
        }
    }
}

2 references
static bool TestFile(string content)
{
    var thread1 = new Thread(() =>
    {
        Write(content);
        Thread.Yield();
    });
    var thread2 = new Thread(() => Environment.Exit(1));
    thread1.Start();
    thread2.Start();

    return true;
}
```



```
    if (args.Length > 0)
    {
        HijackMethod(
            typeof(FileStream).GetMethod("WriteFileNative", BindingFlags.NonPublic | BindingFlags.Instance),
            typeof(Program).GetMethod(nameof(Hijacked), BindingFlags.Instance | BindingFlags.Public)
        );
        TestFile(content);
    }
    else
    {
        while (true)
        {
            var process = Process.Start(new ProcessStartInfo
            {
                Arguments = "slave",
                RedirectStandardOutput = true,
                FileName = "FinalizerRaceCondition.exe",
                UseShellExecute = false,
                CreateNoWindow = false,
                WindowStyle = ProcessWindowStyle.Normal
            });
            process.WaitForExit();
            var output = process.StandardOutput.ReadToEnd();

            if (new FileInfo("0").Length - 3 > 0 || !string.IsNullOrEmpty(output))
            {
                Console.WriteLine(output);
                Debugger.Launch();
                Debugger.Break();
                Thread.Sleep(100000000);
            }
            Thread.Sleep(1);
        }
    }
}

1 reference
public unsafe int Hijacked(SafeFileHandle handle, byte[] bytes, int offset, int co
{
    if (stacktrace != null)
    {
        Console.WriteLine("Current stacktrace");
        Console.WriteLine(Environment.StackTrace);
        Console.WriteLine("Other stacktrace");
        Console.WriteLine(stacktrace);
    }

    stacktrace = Environment.StackTrace;
    Thread.Yield();

    hr = 0;
    return 3;
}
```

C:\Windows\system32\cmd.exe

```
Current stacktrace
   at System.Environment.GetStackTrace(Exception e, Boolean needFileInfo)
   at System.Environment.get_StackTrace()
   at FinalizerRaceCondition.Program.Hijacked(SafeFileHandle handle, Byte[] bytes, Int32 offset, Int32 count, NativeOverlapped* ove
rlapped, Int32& hr)
   at System.IO.FileStream.WriteCore(Byte[] buffer, Int32 offset, Int32 count)
   at System.IO.FileStream.FlushWrite(Boolean calledFromFinalizer)
   at System.IO.FileStream.Dispose(Boolean disposing)
   at System.IO.FileStream.Finalize()
Other stacktrace
   at System.Environment.GetStackTrace(Exception e, Boolean needFileInfo)
   at System.Environment.get_StackTrace()
   at FinalizerRaceCondition.Program.Hijacked(SafeFileHandle handle, Byte[] bytes, Int32 offset, Int32 count, NativeOverlapped* ove
rlapped, Int32& hr) in C:\Users\afish\Desktop\msp_windowsinternals\FinalizerRaceCondition\Program.cs:line 122
   at System.IO.FileStream.WriteCore(Byte[] buffer, Int32 offset, Int32 count)
   at System.IO.FileStream.FlushInternalBuffer()
   at System.IO.FileStream.Flush(Boolean flushToDisk)
   at System.IO.FileStream.Flush()
   at System.IO.StreamWriter.Flush(Boolean flushStream, Boolean flushEncoder)
   at System.IO.StreamWriter.Dispose(Boolean disposing)
   at System.IO.TextWriter.Dispose()
   at FinalizerRaceCondition.Program.Write(String content) in C:\Users\afish\Desktop\msp_windowsinternals\FinalizerRaceCondition\Pr
ogram.cs:line 68
   at FinalizerRaceCondition.Program.<>c__DisplayClass3_0.<TestFile>b__0() in C:\Users\afish\Desktop\msp_windowsinternals\Finalizer
RaceCondition\Program.cs:line 50
   at System.Threading.ThreadHelper.ThreadStart_Context(Object state)
   at System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object state, Bool
ean preserveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state, Boolean pres
erveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state)
   at System.Threading.ThreadHelper.ThreadStart()
```

Exceptions in async

VOID METHOD

One cannot await *void* method (sure?).

Exception is propagated but it is not deterministic.

Use *AsyncContext* from *AsyncEx* library.

TASK METHOD

Exception is stored in the *Task*.

You can also await the method and have the exception propagated.

When chaining in parent-child hierarchy (*TaskCreationOptions.AttachedToParent*) we may miss exceptions, even in *AggregatedException*.

If there is an unobserved exception, it is raised by finalizer thread in *UnobservedTaskException* event where it can be cleared. If not cleared, the process dies (.NET 4) or the exception is suppressed (.NET 4.5).

```
1  using System;
2  using System.Threading;
3  using System.Threading.Tasks;
4
5  namespace ExceptionInAsync
6  {
7
8      class Program
9      {
10         static int Id = 1;
11
12         static async void Throw()
13         //static async Task Throw()
14         {
15             await Task.Delay(300);
16             throw new Exception("I am throwing an exception: " + (Id++));
17         }
18
19         static void Main(string[] args)
20         {
21             // Observe that void method propagates the exception
22             // Task method does not
23
24             try
25             {
26                 Throw();
27                 //Thread.Sleep(600);
28                 Throw();
29             }
30             catch (Exception e)
31             {
32                 Console.WriteLine("Handling " + e);
33             }
34
35             Console.WriteLine("After try");
36             Thread.Sleep(900);
37             Console.WriteLine("After sleep");
38             Console.WriteLine("Done");
39         }
40     }
41 }
42
```

C:\WINDOWS\system32\cmd.exe

```
After try

Unhandled Exception:
Unhandled Exception: System.Exception: I am throwing an exception: 1
   at ExceptionInAsync.Program.<Throw>d__1.MoveNext() in C:\Users\adafurma\Desktop\msp_windowsinternals\ExceptionInAsync
\Program.cs:line 16
--- End of stack trace from previous location where exception was thrown ---
   at System.Runtime.CompilerServices.AsyncMethodBuilderCore.<>c.<ThrowAsync>b__6_1(Object state)
   at System.Threading.QueueUserWorkItemCallback.WaitCallback_Context(Object state)
   at System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object
state, Boolean preserveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state, B
oolean preserveSyncCtx)
   at System.Threading.QueueUserWorkItemCallback.System.Threading.IThreadPoolWorkItem.ExecuteWorkItem()
   at System.Threading.ThreadPoolWorkQueue.Dispatch()
   at System.Threading._ThreadPoolWaitCallback.PerformWaitCallback()
System.Exception: I am throwing an exception: 2
   at ExceptionInAsync.Program.<Throw>d__1.MoveNext() in C:\Users\adafurma\Desktop\msp_windowsinternals\ExceptionInAsync
\Program.cs:line 16
--- End of stack trace from previous location where exception was thrown ---
   at System.Runtime.CompilerServices.AsyncMethodBuilderCore.<>c.<ThrowAsync>b__6_1(Object state)
   at System.Threading.QueueUserWorkItemCallback.WaitCallback_Context(Object state)
   at System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object
state, Boolean preserveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state, B
oolean preserveSyncCtx)
   at System.Threading.QueueUserWorkItemCallback.System.Threading.IThreadPoolWorkItem.ExecuteWorkItem()
   at System.Threading.ThreadPoolWorkQueue.Dispatch()
   at System.Threading._ThreadPoolWaitCallback.PerformWaitCallback()
After sleep
Done
Press any key to continue . . .
```

```
1  using System;
2  using System.Threading;
3  using System.Threading.Tasks;
4
5  namespace ExceptionInAsync
6  {
7
8      class Program
9      {
10         static int Id = 1;
11
12         static async void Throw()
13         //static async Task Throw()
14         {
15             await Task.Delay(300);
16             throw new Exception("I am throwing an exception: " + (Id++));
17         }
18
19         static void Main(string[] args)
20         {
21             // Observe that void method propagates the exception
22             // Task method does not
23
24             try
25             {
26                 Throw();
27                 Thread.Sleep(600);
28                 Throw();
29             }
30             catch (Exception e)
31             {
32                 Console.WriteLine("Handling " + e);
33             }
34
35             Console.WriteLine("After try");
36             Thread.Sleep(900);
37             Console.WriteLine("After sleep");
38             Console.WriteLine("Done");
39         }
40     }
41 }
42
```

C:\WINDOWS\system32\cmd.exe

```
Unhandled Exception: System.Exception: I am throwing an exception: 1
   at ExceptionInAsync.Program.<Throw>d__1.MoveNext() in C:\Users\adafurma\Desktop\msp_windowsinternals\ExceptionInAsync
\Program.cs:line 16
--- End of stack trace from previous location where exception was thrown ---
   at System.Runtime.CompilerServices.AsyncMethodBuilderCore.<>c.<ThrowAsync>b__6_1(Object state)
   at System.Threading.QueueUserWorkItemCallback.WaitCallback_Context(Object state)
   at System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object
state, Boolean preserveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state, B
oolean preserveSyncCtx)
   at System.Threading.QueueUserWorkItemCallback.System.Threading.IThreadPoolWorkItem.ExecuteWorkItem()
   at System.Threading.ThreadPoolWorkQueue.Dispatch()
   at System.Threading._ThreadPoolWaitCallback.PerformWaitCallback()
After try
Press any key to continue . . .
```

```
1  using System;
2  using System.Threading;
3  using System.Threading.Tasks;
4
5  namespace ExceptionInAsync
6  {
7
8      class Program
9      {
10         static int Id = 1;
11
12         //static async void Throw()
13         static async Task Throw()
14         {
15             await Task.Delay(300);
16             throw new Exception("I am throwing an exception: " + (Id++));
17         }
18
19         static void Main(string[] args)
20         {
21             // Observe that void method propagates the exception
22             // Task method does not
23
24             try
25             {
26                 Throw();
27                 //Thread.Sleep(600);
28                 Throw();
29             }
30             catch (Exception e)
31             {
32                 Console.WriteLine("Handling " + e);
33             }
34
35             Console.WriteLine("After try");
36             Thread.Sleep(900);
37             Console.WriteLine("After sleep");
38             Console.WriteLine("Done");
39         }
40     }
41 }
42
```

C:\WINDOWS\system32\cmd.exe

```
After try
After sleep
Done
Press any key to continue . . .
```

Exceptions in other threads

Unhandled exception kills the application in most cases.

If it happens on a thread pool it is held until awaiting and then propagated if possible (thrown out of band for *async void*).

Catching unhandled exception with *AppDomain.CurrentDomain.UnhandledException* doesn't stop the application from terminating.

ThreadAbortException or *AppDomainUnloadedException* **do not** kill the application.

In .NET 1 it was different:

- Exception on a thread pool was printed to the console and the thread was returned to the pool.
- Exception in other thread was printed to the console and the thread was terminated.
- Exception on the finalizer was printed to the console and finalizer was still working.
- Exception on the main thread resulted in application termination.

Hijacking thread creation

Program.cs

ThreadExceptionHandler | ThreadExceptionHandler.Program | Main(string[] args)

```
102                 }
103             }
104
105             public static ThreadStart ModifyHandler(object _, ThreadStart threadStart)
106             {
107                 return () =>
108                 {
109                     try
110                     {
111                         threadStart();
112                     }
113                     catch (Exception e)
114                     {
115                         Console.WriteLine("Handling!");
116                         Console.WriteLine(e);
117                     }
118                 };
119             }
120         }
121
122         class Program
123         {
124             static void Main(string[] args)
125             {
126                 ThreadHandler.EnableHandling();
127
128                 MakeUnhandled();
129
130                 Thread.Sleep(TimeSpan.FromSeconds(3));
131
132                 Console.WriteLine("All done");
133             }
134
135             private static void MakeUnhandled()
136             {
137                 ThreadStart lambda = () =>
138                 {
139                     Console.WriteLine("Running in new thread");
140                     throw new Exception("This is unhandled!");
141                 };
142                 var thread = new Thread(lambda);
143                 thread.Start();
144             }
145         }
146     }
147
```

C:\WINDOWS\system32\cmd.exe

```
Callback modifier caller: 30B24C0
Callback modifier: 1770888
Constructor: 72991A34
Running in new thread
Handling!
System.Exception: This is unhandled!
   at ThreadExceptionHandler.Program.<>c.<MakeUnhandled>b__1_0() in C:\Users\adafurma\Desktop\msp_windowsinternals\Threa
dExceptionHandler\Program.cs:line 140
   at ThreadExceptionHandler.ThreadHandler.<>c__DisplayClass6_0.<ModifyHandler>b__0() in C:\Users\adafurma\Desktop\msp_w
indowsinternals\ThreadExceptionHandler\Program.cs:line 111
All done
Press any key to continue . . .
```

Constrained Executed Region

Area of code in which the CLR is constrained from throwing out-of-band exceptions that would prevent the code from executing in its entirety.

PrepareConstrainedRegions and try block must be used. Code must be marked with *ReliabilityContractAttribute*. Code is compiled, 48kB on the stack are prepared (the average size of a method).

Not allowed:

- Explicit allocation (including boxing).
- Acquiring a *lock*.
- Calling unprepared methods virtually. Calling methods with weak or nonexistent reliability contract.
- Using multidimensional arrays, reflection, security checks, serialization, many more.

CriticalFinalizerObject can be used to guarantee execution of the finalizer even for rude aborts. It is used by *SafeHandle* instances.

RuntimeHelpers.ExecuteCodeWithGuaranteedCleanup guarantees execution of the cleanup method (by using SEH).

Constrained Executed Region

```
CompilerServices.RuntimeServices.PrepareConstrainedRegions();
try {

} catch {

   // CER is here

} finally {

   // CER is here

}
```

Even more implementation details

Process: [7248] ThrowInCsharp.exe | Lifecycle Events | Thread: [7300] Main Thread | Stack Frame: ThrowInCsharp.Program.Main

Program.cs

ThrowInCsharp | ThrowInCsharp.Program | Main(string[] args)

```
using System;

namespace ThrowInCsharp
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            try
            {
                throw new Exception("5");
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }
    }
}
```

Disassembly

Address: ThrowInCsharp.Program.Main(string[])

Viewing Options

```
00007FFEB178149E  call        00007FFF1139D2D0
00007FFEB17814A3  nop
            try
            {
00007FFEB17814A4  nop
                throw new Exception("5");
00007FFEB17814A5  mov         rcx,7FFEB1804170h
00007FFEB17814AF  call        00007FFF11277380
00007FFEB17814B4  mov         qword ptr [rbp+38h],rax
00007FFEB17814B8  mov         ecx,1
00007FFEB17814BD  mov         rdx,7FFEB18216E0h
00007FFEB17814C7  call        00007FFF1139FFB0
00007FFEB17814CC  mov         qword ptr [rbp+28h],rax
00007FFEB17814D0  mov         rdx,qword ptr [rbp+28h]
00007FFEB17814D4  mov         rcx,qword ptr [rbp+38h]
00007FFEB17814D8  call        00007FFEB1774C18
00007FFEB17814DD  mov         rcx,qword ptr [rbp+38h]
00007FFEB17814E1  call        00007FFF111EA880
00007FFEB17814E6  int         3
        }
00007FFEB17814E7  nop
00007FFEB17814E8  lea         rsp,[rbp+50h]
00007FFEB17814EC  pop         rsi
00007FFEB17814ED  pop         rdi
00007FFEB17814EE  pop         rbp
00007FFEB17814EF  ret
        {
00007FFEB17814F0  push        rbp
00007FFEB17814F1  push        rdi
00007FFEB17814F2  push        rsi
00007FFEB17814F3  sub         rsp,30h
00007FFEB17814F7  mov         rbp,qword ptr [rcx+20h]
00007FFEB17814FB  mov         qword ptr [rsp+20h],rbp
            }
            catch (Exception e)
00007FFEB1781500  mov         qword ptr [rbp+30h],rdx
00007FFEB1781504  mov         rcx,qword ptr [rbp+30h]
00007FFEB1781508  mov         qword ptr [rbp+40h],rcx
            {
00007FFEB178150C  nop
                Console.WriteLine(e);
00007FFEB178150D  mov         rcx,qword ptr [rbp+40h]
00007FFEB1781511  call        00007FFEB1781458
00007FFEB1781516  nop
```

Pid 7248 - WinDbg:10.0.18362.1 AMD64

File Edit View Debug Window Help

Command

```
C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 11:
00007ffe`b17814a5 cc              int     3
00007ffe`b17814a6 b9704180b1      mov     ecx,0B1804170h
00007ffe`b17814ab fe              ???
00007ffe`b17814ac 7f00            jg      00007ffe`b17814ae
00007ffe`b17814ae 00e8            add     al,ch
00007ffe`b17814b0 cc              int     3
00007ffe`b17814b1 5e              pop     rsi
00007ffe`b17814b2 af              scas    dword ptr [rdi]
00007ffe`b17814b3 5f              pop     rdi
00007ffe`b17814b4 48894538        mov     qword ptr [rbp+38h],rax
00007ffe`b17814b8 b901000000      mov     ecx,1
00007ffe`b17814bd 48bae01682b1fe7f0000 mov rdx,7FFEB18216E0h
00007ffe`b17814c7 e8e4eac15f      call    coreclr!JIT_StrCns (00007fff`1139ffb0)
00007ffe`b17814cc 48894528        mov     qword ptr [rbp+28h],rax
00007ffe`b17814d0 488b5528        mov     rdx,qword ptr [rbp+28h]
00007ffe`b17814d4 488b4d38        mov     rcx,qword ptr [rbp+38h]
00007ffe`b17814d8 e83b37ffff      call    00007ffe`b1774e18 (System.Exception.ctor(System.String), mdToken: 000000000600036E)
00007ffe`b17814dd 488b4d38        mov     rcx,qword ptr [rbp+38h]
00007ffe`b17814e1 e89a93a65f      call    coreclr!IL_Throw (00007fff`111ea880)
00007ffe`b17814e6 cc              int     3

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 17:
00007ffe`b17814e7 90              nop
00007ffe`b17814e8 488d6550        lea     rsp,[rbp+50h]
00007ffe`b17814ec 5e              pop     rsi
00007ffe`b17814ed 5f              pop     rdi
00007ffe`b17814ee 5d              pop     rbp
00007ffe`b17814ef c3              ret

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 8:
00007ffe`b17814f0 55              push    rbp
00007ffe`b17814f1 57              push    rdi
00007ffe`b17814f2 56              push    rsi
00007ffe`b17814f3 4883ec30        sub     rsp,30h
00007ffe`b17814f7 488b6920        mov     rbp,qword ptr [rcx+20h]
00007ffe`b17814fb 48896c2420      mov     qword ptr [rsp+20h],rbp

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 13:
00007ffe`b1781500 48895530        mov     qword ptr [rbp+30h],rdx
00007ffe`b1781504 488b4d30        mov     rcx,qword ptr [rbp+30h]
00007ffe`b1781508 48894d40        mov     qword ptr [rbp+40h],rcx

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 14:
00007ffe`b178150c 90              nop

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 15:
00007ffe`b178150d 488b4d40        mov     rcx,qword ptr [rbp+40h]
00007ffe`b1781511 e842ffffff      call    00007ffe`b1781458 (System.Console.WriteLine(System.Object), mdToken: 0000000006000080)
00007ffe`b1781516 90              nop

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 16:
00007ffe`b1781517 90              nop
00007ffe`b1781518 90              nop
00007ffe`b1781519 488d05c7ffffff  lea     rax,[00007ffe`b17814e7]

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 17:
```

0:000>

Disassembly

Address: ThrowInCsharp.Program.Main(string[])

Viewing Options

```
00007FFEB178149E  call    00007FFF1139D2D0
00007FFEB17814A3  nop
            try
            {
00007FFEB17814A4  nop
                throw new Exception("5");
00007FFEB17814A5  mov     rcx,7FFEB1804170h
00007FFEB17814AF  call    00007FFF11277380
00007FFEB17814B4  mov     qword ptr [rbp+38h],rax
00007FFEB17814B8  mov     ecx,1
00007FFEB17814BD  mov     rdx,7FFEB18216E0h
00007FFEB17814C7  call    00007FFF1139FFB0
00007FFEB17814CC  mov     qword ptr [rbp+28h],rax
00007FFEB17814D0  mov     rdx,qword ptr [rbp+28h]
00007FFEB17814D4  mov     rcx,qword ptr [rbp+38h]
00007FFEB17814D8  call    00007FFEB1774C18
00007FFEB17814DD  mov     rcx,qword ptr [rbp+38h]
00007FFEB17814E1  call    00007FFF111EA880
00007FFEB17814E6  int     3
        }
00007FFEB17814E7  nop
00007FFEB17814E8  lea     rsp,[rbp+50h]
00007FFEB17814EC  pop     rsi
00007FFEB17814ED  pop     rdi
00007FFEB17814EE  pop     rbp
00007FFEB17814EF  ret
        {
00007FFEB17814F0  push    rbp
00007FFEB17814F1  push    rdi
00007FFEB17814F2  push    rsi
00007FFEB17814F3  sub     rsp,30h
00007FFEB17814F7  mov     rbp,qword ptr [rcx+20h]
00007FFEB17814FB  mov     qword ptr [rsp+20h],rbp
            }
            catch (Exception e)
00007FFEB1781500  mov     qword ptr [rbp+30h],rdx
00007FFEB1781504  mov     rcx,qword ptr [rbp+30h]
00007FFEB1781508  mov     qword ptr [rbp+40h],rcx
            {
00007FFEB178150C  nop
                Console.WriteLine(e);
00007FFEB178150D  mov     rcx,qword ptr [rbp+40h]
00007FFEB1781511  call    00007FFEB1781458
00007FFEB1781516  nop
```

100 %

Pid 7248 - WinDbg:10.0.18362.1 AMD64

File Edit View Debug Window Help

Command

```
00007ffe`b17814a4 90              nop

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 11:
00007ffe`b17814a5 cc              int     3
00007ffe`b17814a6 b9704180b1      mov     ecx,0B1804170h
00007ffe`b17814ab fe              ???
00007ffe`b17814ac 7f00            jg      00007ffe`b17814ae
00007ffe`b17814ae 00e8            add     al,ch
00007ffe`b17814b0 cc              int     3
00007ffe`b17814b1 5e              pop     rsi
00007ffe`b17814b2 af              scas    dword ptr [rdi]
00007ffe`b17814b3 5f              pop     rdi
00007ffe`b17814b4 48894538        mov     qword ptr [rbp+38h],rax
00007ffe`b17814b8 b901000000      mov     ecx,1
00007ffe`b17814bd 48bae01682b1fe7f0000 mov rdx,7FFEB18216E0h
00007ffe`b17814c7 e8e4eac15f      call    coreclr!JIT_StrCns (00007fff`1139ffb0)
00007ffe`b17814cc 48894528        mov     qword ptr [rbp+28h],rax
00007ffe`b17814d0 488b5528        mov     rdx,qword ptr [rbp+28h]
00007ffe`b17814d4 488b4d38        mov     rcx,qword ptr [rbp+38h]
00007ffe`b17814d8 e83b37ffff      call    00007ffe`b1774c18 (System.Exception..ctor(System.String), mdToken: 000000000600036E)
00007ffe`b17814dd 488b4d38        mov     rcx,qword ptr [rbp+38h]
00007ffe`b17814e1 e89a93a65f      call    coreclr!IL_Throw (00007fff`111ea880)
00007ffe`b17814e6 cc              int     3

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 17:
00007ffe`b17814e7 90              nop
00007ffe`b17814e8 488d6550        lea     rsp,[rbp+50h]
00007ffe`b17814ec 5e              pop     rsi
00007ffe`b17814ed 5f              pop     rdi
00007ffe`b17814ee 5d              pop     rbp
00007ffe`b17814ef c3              ret

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 8:
00007ffe`b17814f0 55              push    rbp
00007ffe`b17814f1 57              push    rdi
00007ffe`b17814f2 56              push    rsi
00007ffe`b17814f3 4883ec30        sub     rsp,30h
00007ffe`b17814f7 488b6920        mov     rbp,qword ptr [rcx+20h]
00007ffe`b17814fb 48896c2420      mov     qword ptr [rsp+20h],rbp

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 13:
00007ffe`b1781500 48895530        mov     qword ptr [rbp+30h],rdx
00007ffe`b1781504 488b4d30        mov     rcx,qword ptr [rbp+30h]
00007ffe`b1781508 48894d40        mov     qword ptr [rbp+40h],rcx

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 14:
00007ffe`b178150c 90              nop

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 15:
00007ffe`b178150d 488b4d40        mov     rcx,qword ptr [rbp+40h]
00007ffe`b1781511 e842ffffff      call    00007ffe`b1781458 (System.Console.WriteLine(System.Object), mdToken: 0000000006000080)
00007ffe`b1781516 90              nop

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 16:
00007ffe`b1781517 90              nop
00007ffe`b1781518 90              nop
00007ffe`b1781519 488d05c7ffffff  lea     rax,[00007ffe`b17814e7]

C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCsharp\Program.cs @ 17:
```

0:000>

Disassembly

Address: ThrowInCsharp.Program.Main(string[])

Viewing Options

```
00007FFEB178149E  call    00007FFF1139D2D0
00007FFEB17814A3  nop
            try
            {
00007FFEB17814A4  nop
                throw new Exception("E");
00007FFEB17814A5  mov     rcx,7FFEB1804170h
00007FFEB17814AF  call    00007FFF11277380
00007FFEB17814B4  mov     qword ptr [rbp+38h],rax
00007FFEB17814B8  mov     ecx,1
00007FFEB17814BD  mov     rdx,7FFEB18216E0h
00007FFEB17814C7  call    00007FFF1139FFB0
00007FFEB17814CC  mov     qword ptr [rbp+28h],rax
00007FFEB17814D0  mov     rdx,qword ptr [rbp+28h]
00007FFEB17814D4  mov     rcx,qword ptr [rbp+38h]
00007FFEB17814D8  call    00007FFEB1774C18
00007FFEB17814DD  mov     rcx,qword ptr [rbp+38h]
00007FFEB17814E1  call    00007FFF111EA880
00007FFEB17814E6  int     3
        }
00007FFEB17814E7  nop
00007FFEB17814E8  lea     rsp,[rbp+50h]
00007FFEB17814EC  pop     rsi
00007FFEB17814ED  pop     rdi
00007FFEB17814EE  pop     rbp
00007FFEB17814EF  ret
        {
00007FFEB17814F0  push    rbp
00007FFEB17814F1  push    rdi
00007FFEB17814F2  push    rsi
00007FFEB17814F3  sub     rsp,30h
00007FFEB17814F7  mov     rbp,qword ptr [rcx+20h]
00007FFEB17814FB  mov     qword ptr [rsp+20h],rbp
            }
            catch (Exception e)
00007FFEB1781500  mov     qword ptr [rbp+30h],rdx
00007FFEB1781504  mov     rcx,qword ptr [rbp+30h]
00007FFEB1781508  mov     qword ptr [rbp+40h],rcx
            {
00007FFEB178150C  nop
                Console.WriteLine(e);
00007FFEB178150D  mov     rcx,qword ptr [rbp+40h]
00007FFEB1781511  call    00007FFEB1781458
00007FFEB1781516  nop
```

100 %

Process: [4392] ThrowInCpp.exe | Lifecycle Events | Thread: [7300] Main Thread | Stack Frame: main

ThrowInCpp.cpp

ThrowInCpp | (Global Scope)

```
1   #include <iostream>
2
3   int main()
4   {
5       try {
6           throw 5;
7       }
8       catch (int& e) {
9           std::cout << "Exception: " << e;
10      }
11  }
12
```

100 % | No issues found | Ln: 12 | Ch: 1 | SPC | CRLF

Disassembly

Address: main(void)

Viewing Options

```
0006252E  mov         dword ptr [ebp-10h],esp
00062531  mov         ecx,offset _FF3693CC_ThrowInCpp@cpp (06E026h)
00062536  call        @__CheckForDebuggerJustMyCode@4 (06128Ah)
    try {
0006253B  mov         dword ptr [ebp-4],0
        throw 5;
00062542  mov         dword ptr [ebp-0E4h],5
0006254C  push        offset __TI1H (06B1ECh)
00062551  lea         eax,[ebp-0E4h]
00062557  push        eax
00062558  call        __CxxThrowException@8 (06141Ah)
    }
    catch (int& e) {
        std::cout << "Exception: " << e;
0006255D  mov         esi,esp
0006255F  mov         eax,dword ptr [ebp-18h]
00062562  mov         ecx,dword ptr [eax]
00062564  push        ecx
00062565  push        offset string "Exception: " (069B30h)
0006256A  mov         edx,dword ptr [_imp_?cout@std@@3V?$basic_ostream@DU?$char_traits(
00062570  push        edx
00062571  call        std::operator<<<std::char_traits<char> > (06121Ch)
00062576  add         esp,8
00062579  mov         ecx,eax
0006257B  call        dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> :
00062581  cmp         esi,esp
00062583  call        __RTC_CheckEsp (061294h)
    }
00062588  mov         eax,offset $LN7 (062597h)
0006258D  ret
0006258E  mov         dword ptr [ebp-4],0FFFFFFFFh
00062595  jmp         $LN7+7h (06259Eh)
$LN7:
00062597  mov         dword ptr [ebp-4],0FFFFFFFFh
}
0006259E  jmp         $LN7+0Bh (0625A2h)
000625A0  jmp         $LN7+0Dh (0625A4h)
000625A2  xor         eax,eax
000625A4  push        edx
000625A5  mov         ecx,ebp
}
000625A7  push        eax
000625A8  lea         edx,ds:[625D4h]
000625AF  call        @_RTC_CheckStackVars@8 (0612B7h)
```

100 %

Call Stack | Breakpoints | Exception Settings | Command Window | Immediate Window | Output | Autos | Locals | Watch 1 | Threads | Tasks

Ready

Throw in C++

(17b8.4d0): **C++ EH exception - code e06d7363 (first chance)**

First chance exceptions are reported before any exception handling.

This exception may be expected and handled.

eax=001df6d8 ebx=0031b000 ecx=00000003 edx=00000000 esi=000613c5 edi=001df868

eip=76ee3db2 esp=001df6d8 ebp=001df734 iopl=0 nv up ei pl nz ac pe nc

cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000216

KERNELBASE!RaiseException+0x62:

76ee3db2 8b4c2454 mov ecx,dword ptr [esp+54h] ss:002b:001df72c=6a31729e

Throw in C++

0:000> kb

ChildEBP RetAddr Args to Child

00 001df734 716b9e80 e06d7363 00000001 00000003 KERNELBASE!RaiseException+0x62

01 001df770 0006255d 001df794 0006b1ec cf84f650
VCRUNTIME140D!_CxxThrowException+0xa0
[d:\agent_work\3\s\src\vctools\crt\vcruntime\src\eh\throw.cpp @ 75]

02 001df878 00062e03 00000001 00744b18 00747250 ThrowInCpp!main+0x6d
[C:\Users\afish\Desktop\msp_windowsinternals\ThrowInCpp\ThrowInCpp.cpp @ 9]

Throw in C++

```
0:000> u VCRUNTIME140D!_CxxThrowException+0xa0-6
VCRUNTIME140D!_CxxThrowException+0x9a [d:\agent\_work\3\s\src\vctools\crt\vcruntime\src\eh\throw.cpp @ 74]:
716b9e7a ff1558a06c71    call    dword ptr [VCRUNTIME140D!_imp__RaiseException (716ca058)]
716b9e80 8be5            mov     esp,ebp
716b9e82 5d              pop     ebp
716b9e83 c20800          ret     8
0:000> u 716ca058
VCRUNTIME140D!_imp__RaiseException:
716ca058 40              inc     eax
716ca059 7dcc            jge     VCRUNTIME140D!_imp__IsProcessorFeaturePresent+0x3 (716ca027)
716ca05b 7400            je      VCRUNTIME140D!_imp__EnterCriticalSection+0x1 (716ca05d)
716ca05d 60              pushad
716ca05e 4f              dec     edi
716ca05f 77b0            ja      VCRUNTIME140D!_imp__UnhandledExceptionFilter+0x1 (716ca011)
716ca061 98              cwde
716ca062 cc              int     3
```

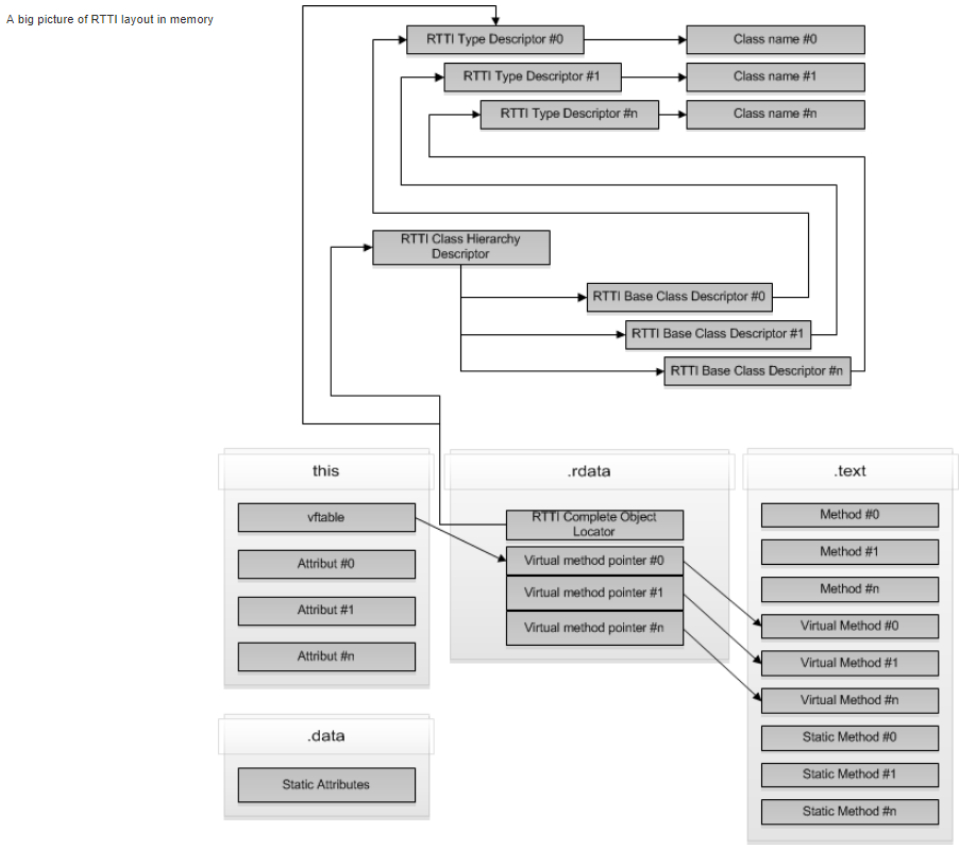
Exceptions as seen in WinDBG

Handling exception by code:
- sxe CODE

C++ Exception:
- Code 0xE06D7363.
- Parameter 1 — pointer to the thrown object.
- Parameter 2 — pointer to object description.

Delphi Exception:
- Code 0x0EEDFADE.
- Parameter 1 — pointer to the thrown object.
- Parameter 2 — pointer to object description.

CLR Exception:
- Code 0xE0434F4D.
- Use SOS !StopOnException to specify exception type to break on.



https://blog.quarkslab.com/visual-c-rtti-inspection.html

SEH

X86

Chain of handlers stored in *fs:[0]* register.

Handlers stored on the stack.

Overhead because of storing the handler on the stack with every method call.

Dangerous because of buffer overflow attacks.

Use !exchain in WinDBG to see the handlers.

You can use *SAFESEH* to emit the handler tables.

X64

Table-based.

Source code is compiled and the table with handlers is created.

Processing exception is a little harder but there is smaller overhead when calling the function.

Tables are stored in PE, no data on the stack.

_IMAGE_RUNTIME_FUNCTION_ENTRY structure in .pdata section with begin/end addresses and pointer to an exception handler.

```
6   EXCEPTION_DISPOSITION
7   __cdecl
8   _except_handler(
9       struct _EXCEPTION_RECORD *ExceptionRecord,
10      void * EstablisherFrame,
11      struct _CONTEXT *ContextRecord,
12      void * DispatcherContext)
13  {
14      puts("Hello from an exception handler");
15
16      // Change EIP in the context record to skip invalid instruction (emulating jump after catch)
17      ContextRecord->Eip += 3;
18
19      // Tell the OS to go forward
20      return ExceptionContinueExecution;
21  }
22
23  int main()
24  {
25      DWORD handler = (DWORD)_except_handler;
26
27      puts("Registering SEH!");
28
29      __asm
30      {                           // Build EXCEPTION_REGISTRATION record:
31          push    handler         // Address of handler function
32          push    FS : [0]        // Address of previous handler
33          mov     FS : [0], ESP   // Install new EXECEPTION_REGISTRATION
34      }
35
36      puts("Dividing by zero!");
37
38      __asm
39      {
40          mov eax, 0              // Zero out EAX
41          mov [eax], 1            // Write to EAX to deliberately cause a fault
42      }
43
44      puts("After division!");
45
46      __asm
47      {                           // Remove our EXECEPTION_REGISTRATION record
48          mov     eax, [ESP]      // Get pointer to previous record
49          mov     FS : [0], EAX   // Install previous record
50          add     esp, 8          // Clean our EXECEPTION_REGISTRATION off stack
51      }
52
53      return 0;
54  }
```

C:\WINDOWS\system32\cmd.exe

```
Registering SEH!
Dividing by zero!
Hello from an exception handler
After division!
Press any key to continue . . .
```

VEH, VCH

Internals very undocumented.

To find all registered handlers in x86 just iterate over all possible addresses and try to remove them.

For x64 go through kernel debugging.

VCH is called when VEH or SEH returns *EXCEPTION_CONTINUE_EXECUTION*.

It is also also called if SEH validation fails.

IL Metadata

Exception tables are located right after the method IL.

Filter just returns zero or one indicating whether the following handler is involed in exception handling.

Guarded block (*try*) with *finally* or *fault* cannot have other handler (*catch*).

IL doesn't enforce catch subtype ordering, this is done by the C# compiler.

Can you always add „redundant nop" instruction?

ConsoleApplication1 - Microsoft Visual Studio

File Edit View Project Build Debug Team CodeRush Tools

Debug | Any CPU

Program.cs

ConsoleApplication1

```
1   using System;
2
3   namespace ConsoleApplication1
4   {
5       class Program
6       {
7           static void Main(string[] args)
8           {
9               try
10              {
11                  Console.WriteLine("Try");
12              }catch(InvalidOperationException e)
13              {
14                  Console.WriteLine("Catch 1");
15              }
16              catch(Exception e)
17              {
18                  Console.WriteLine("Catch 2");
19              }
20              finally
21              {
22                  Console.WriteLine("Finally");
23              }
24          }
25      }
26  }
27
```

ConsoleApplication1.Program::Main : void(string[])

Find Find Next

```
.method /*06000001*/ private hidebysig static
        void  Main(string[] args) cil managed
{
  .entrypoint
  // Code size       65 (0x41)
  .maxstack  1
  .locals /*11000001*/ init ([0] class [mscorlib/*23000001*/]System.InvalidOperationException/*01000011*/ e,
           [1] class [mscorlib/*23000001*/]System.Exception/*01000012*/ V_1)
  IL_0000:  nop
  IL_0001:  nop
  IL_0002:  ldstr      "Try" /* 70000001 */
  IL_0007:  call       void [mscorlib/*23000001*/]System.Console/*01000013*/::WriteLine(string) /* 0A00000F */
  IL_000c:  nop
  IL_000d:  nop
  IL_000e:  leave.s    IL_0030
  IL_0010:  stloc.0
  IL_0011:  nop
  IL_0012:  ldstr      "Catch 1" /* 70000009 */
  IL_0017:  call       void [mscorlib/*23000001*/]System.Console/*01000013*/::WriteLine(string) /* 0A00000F */
  IL_001c:  nop
  IL_001d:  nop
  IL_001e:  leave.s    IL_0030
  IL_0020:  stloc.1
  IL_0021:  nop
  IL_0022:  ldstr      "Catch 2" /* 70000019 */
  IL_0027:  call       void [mscorlib/*23000001*/]System.Console/*01000013*/::WriteLine(string) /* 0A00000F */
  IL_002c:  nop
  IL_002d:  nop
  IL_002e:  leave.s    IL_0030
  IL_0030:  leave.s    IL_0040
  IL_0032:  nop
  IL_0033:  ldstr      "Finally" /* 70000029 */
  IL_0038:  call       void [mscorlib/*23000001*/]System.Console/*01000013*/::WriteLine(string) /* 0A00000F */
  IL_003d:  nop
  IL_003e:  nop
  IL_003f:  endfinally
  IL_0040:  ret
  IL_0041:
  // Exception count 3
  .try IL_0001 to IL_0010 catch [mscorlib/*23000001*/]System.InvalidOperationException/*01000011*/ handler IL_0010 to IL_0020
  .try IL_0001 to IL_0010 catch [mscorlib/*23000001*/]System.Exception/*01000012*/ handler IL_0020 to IL_0030
  .try IL_0001 to IL_0032 finally handler IL_0032 to IL_0040
} // end of method Program::Main
```

Try Performance

```
[Benchmark(Baseline = true)]
public long Inlining1()
{
    long Helper()
    {
        return 1;
    }

    long result = 0;
    for(int i = 0; i < 100; ++i)
    {
        result += Helper();
    }

    return result;
}

[Benchmark]
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public long Inlining2()
{
    long Helper()
    {
        try { return 1; }
        catch { return 1; }
    }

    long result = 0;
    for (int i = 0; i < 100; ++i)
    {
        result += Helper();
    }

    return result;
}
```

C:\Windows\system32\cmd.exe

Method	Mean	Error	StdDev	Median	Scaled	ScaledSD
Inlining1	41.83 ns	0.8811 ns	0.7358 ns	41.75 ns	1.00	0.00
Inlining2	221.31 ns	4.7157 ns	11.1154 ns	216.91 ns	5.29	0.28

Try Performance

```
private const int N = 93;

[Benchmark(Baseline = true)]
public long Fibonacci1()
{
    long a = 0, b = 0, c = 1;
    for (int i = 1; i < N; i++)
    {
        a = b;
        b = c;
        c = a + b;
    }
    return c;
}
```

C:\Windows\system32\cmd.exe

Method	Mean	Error	StdDev	Scaled	ScaledSD
Fibonacci1	127.0 ns	2.729 ns	5.875 ns	1.00	0.00
Fibonacci2	159.2 ns	4.432 ns	7.527 ns	1.26	0.08

```
[Benchmark]
public long Fibonacci2()
{
    long a = 0, b = 0, c = 1;
    try
    {
        for (int i = 1; i < N; i++)
        {
            a = b;
            b = c;
            c = a + b;
        }
    }
    catch { }
    return c;
}
```

```
Fibonacci.Fibonacci1()
    L0000: push ebp
    L0001: mov ebp, esp
    L0003: push edi
    L0004: push esi

    L0005: xor eax, eax
    L0007: xor edx, edx
    L0009: xor ecx, ecx
    L000b: mov esi, 0x1
    L0010: mov edi, 0x1

    L0015: add eax, esi
    L0017: adc edx, ecx
    L0019: inc edi
    L001a: cmp edi, 0x5d
    L001d: jl L002b
    L001f: mov ecx, edx
    L0021: mov esi, eax
    L0023: mov eax, esi
    L0025: mov edx, ecx
    L0027: pop esi
    L0028: pop edi
    L0029: pop ebp
    L002a: ret
    L002b: xchg esi, eax
    L002d: xchg edx, ecx
    L002f: jmp L0015
```

```
Fibonacci.Fibonacci2()
    L0000: push ebp
    L0001: mov ebp, esp
    L0003: push edi
    L0004: push esi
    L0005: sub esp, 0x1c
    L0008: xor eax, eax

    L000a: mov [ebp-0x1c], eax
    L000d: mov [ebp-0x18], eax
    L0010: mov [ebp-0x14], eax
    L0013: mov [ebp-0x10], eax
    L0016: xor eax, eax
    L0018: xor edx, edx
    L001a: mov ecx, 0x1
    L001f: xor esi, esi
    L0021: mov [ebp-0x24], ecx
    L0024: mov [ebp-0x20], esi
    L0027: mov ecx, 0x1
    L002c: mov esi, [ebp-0x24]
    L002f: mov edi, [ebp-0x20]
    L0032: add eax, [ebp-0x24]
    L0035: adc edx, [ebp-0x20]

    L0038: mov [ebp-0x24], eax
    L003b: mov [ebp-0x20], edx
    L003e: inc ecx
    L003f: cmp ecx, 0x5d
    L0042: mov eax, esi
    L0044: mov edx, edi
    L0046: jl L002c
    L0048: jmp L004f
    L004a: call 0x72509369
    L004f: mov eax, [ebp-0x24]
    L0052: mov edx, [ebp-0x20]
    L0055: lea esp, [ebp-0x8]
    L0058: pop esi
    L0059: pop edi
```

lock(locker) { … }

```
01. public static void Lock() {
02.   Monitor.Enter(locker);
03.   nop
04.   try {
05.       ...
06.   } finally {
07.       Monitor.Exit(locker);
08.   }
09. }
```

```
01. public static void Lock() {
02.   try {
03.       Monitor.Enter(locker, ref l)
04.       ...
05.   } finally {
06.       if (l) {
07.           Monitor.Exit(locker);
08.       }
09.   }
10. }
```

Returning in finally

C#, IL — not allowed.

C++ — no finally at all.

Java, SEH, Python, JS… — value returned in finally wins.

But… never return in finally!

```c
void GenerateException(){
    int divideByZero = 0;
    printf("\t\t\tDivided! %d", 1 / divideByZero);
}

int Filter(){
    puts("In filter");
    return EXCEPTION_EXECUTE_HANDLER;
}

void ThirdTry(){
    __try{
        puts("\t\tEntering try 3");
        GenerateException();
        puts("\t\tExiting try 3");
    }
    __finally{
        puts("\t\tIn finally 3");
        return;
    }
}

void SecondTry(){
    __try{
        puts("\tEntering try 2");
        ThirdTry();
        puts("\tExiting try 2");
    }
    __finally{
        puts("\tIn finally 2");
    }
}

void FirstTry(){
    __try{
        puts("Entering try 1");
        SecondTry();
        puts("Exiting try 1");
    }
    __except(Filter()){
        puts("In __except");
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    FirstTry();
    return 0;
}
```

C:\WINDOWS\system32\cmd.exe

```
Entering try 1
        Entering try 2
                Entering try 3
In filter
                In finally 3
        Exiting try 2
        In finally 2
Exiting try 1
Press any key to continue . . .
```

Never return in finally

main.py

```python
def foo():
    try:
        raise Exception("This disappears")
    finally:
        return 42

print(foo())
```

42

</> source code

```java
class Ideone
{
    public static void main (String[] args) throws java.lang.Exception
    {
        System.out.println(foo());
    }

    public static int foo(){
        try{
            throw new RuntimeException("This disappears");
        }finally{
            return 42;
        }
    }
}
```

input Output

Success #stdin #stdout 0.06s 32352KB

42

```javascript
function foo(){
  try{
    throw "This disappears";
  }finally{
    return 42;
  }
}
console.log(foo());
```

42

.NET two-pass exception system

FIRST PASS

Exception is delivered from the top frame to the bottom.

It is stopped as soon as some catch handler (frame) wants to handle the exception.

SECOND PASS

State of each frame is unwind.

Finally and fault are called.

Catch clause in handling frame is executed.

Catching rules

```
01. void Foo() {
02.   try {
03.     throw new Exception("Nope");
04.   } catch (Exception e) {
05.     if (e.Message == "Custom") {
06.         ...
07.     } else {
08.       throw;
09.     }
10.   }
11. }
12.
13. void Main() {
14.     Foo();
15. }
```

```
01. void Foo() {
02.   try {
03.     throw new Exception("Nope");
04.   } catch (Exception e)
05.     when (e.Message == "Custom") {
06.       ...
07.       ...
08.       ...
09.       ...
10.   }
11. }
12.
13. void Main() {
14.     Foo();
15. }
```

C# Exception filters in .NET Framework

CHECKING EXCEPTION TYPE IN THE CODE

```
0:000> !clrstack

00f3eefc 01580537 Handler() [Program.cs @ 11]
00f3eff0 0158049c Main() [Program.cs @ 14]
```

USING EXCEPTION FILTER

```
0:000> !clrstack
008ff47c 00ab0607 Method() [Program.cs @ 04]

00f3eff0 0158049c Main() [Program.cs @ 14]
```

C# Exception filters in .NET Core

CHECKING EXCEPTION TYPE IN THE CODE

```
0:000> !clrstack
00f3eefc 01580537 Handler() [Program.cs @ 11]
00f3f170 70131376 [HelperMethodFrame: 00f3f170]
00f3eefc 01580537 Handler() [Program.cs @ 3]
00f3eff0 0158049c Main() [Program.cs @ 14]
```

USING EXCEPTION FILTER

```
0:000> !clrstack


008ff47c 00ab0607 Method() [Program.cs @ 03]
00f3eff0 0158049c Main() [Program.cs @ 14]
```

Thread.Abort() behavior

.NET FRAMEWORK

Causes *ThreadAbortException*.

Does not interrupt catch and finally blocks.

Exception can be suppressed by calling *Thread.ResetAbort()*.

Effectively it **guarantees nothing** — the thread can catch the exception and carry on.

In .NET 1 exception could be thrown in finally block.

Can accidentally break static constructors.

.NET CORE

Thread.Abort() is not availabe — *PlatformNotSpportedException*.

Thread.Abort() internals

1. Suspend OS thread.
2. Set metadata bit indicating that abort was requested.
3. Add Asynchronous Procedure Call (APC) to the queue. Resume the thread.
4. Thread works again, when it gets to *alertable* state, it executes the APC. It checks the flag and throws the exception.

How to get to *alertable* state?

- IO function, sleep.

What if we never get to the *alertable* state?

- .NET hijacks the thread — it **modifies IP register directly.**

SEHException mapping

STATUS_NO_MEMORY:

- Mapped to *OutOfMemoryException*.

STATUS_ACCESS_VIOLATION:

- Read/write outside of JIT-compiled code —> *AccessViolationException*.
- Read/write inside JIT-compiled code but the address is outside of the null-pointers partition —> *AccessViolationException*.
- If legacyNullReferencePolicy is applied —> *NullReferenceException*.
- If the policy is not applied and the read/write is inside JIT-compiled code and the address is inside null-pointers partition —> *NullReferenceException*.

Otherwise:

- Mapped to *SEHException*.

AccessViolation trickery

```
try {
    Marshal.WriteByte((IntPtr) 1000, 42);
}
catch (AccessViolationException) {
    // Yep, this works!
}
```

Under the hood *WriteByte* tries to write bytes through the pointer.

We write in the null pointer partition, so the *NullReferenceException* is thrown.

It is later handled and converted to *throw new AccessViolationException()*.

It would not be handled if the address was outside of null pointer partition.

```
try {
    Marshal.Copy(new byte[] {42},0, (IntPtr)
1000, bytes.length);
}
catch (AccessViolationException) {
    // Never happens!
}
```

This uses native code which throws *AccessViolationException*.

It cannot be handled by default.

AccessViolation origin

Code / Address	Null pointer partition (< 64 KB)	Other partitions (>= 64 KB)
Managed	NullReferenceException	AccessViolationException
Native	AccessViolationException	AccessViolationException

How does null-check work?

- .NET does not check whether the reference is null or not.
- It just tries to write the memory.
- If the reference is null, Memory Management Unit (MMU) notifies the CPU.
- Exception is then handled by .NET and the check is performed — if this was null pointer partition, NRE is thrown.
- This results in **much faster** positive path.

AccessViolation history

.NET 1:
- No *AccessViolationException*. Reference either correct or null.

.NET 2:
- *AccessViolationException* available and can be handled.

.NET 4:
- Handling requires *HandledProcessCorruptedStateException*.

.NET Core:
- *HandleProcessCorruptedStateException* is ignored. Some exceptions can be restored with configuration switches.

Configuration switches:
- *legacyNullReferenceExceptionPolicy*
  - Brings back .NET 1 behavior — all AV exceptions become NRE.
- *legacyCorruptedStateExceptionPolicy*
  - Brings back .NET 2 behavior — all AV can be handled.
- *COMPlus_legacyCorruptedStateExceptionsPolicy*
  - Some AV exceptions are delivered to the application

Behavior is based on the *TargetFramework* **specified during compilation**, not on the executing platform.

Handling corrupted state in .NET

Before .NET 4 you could just catch access violations exceptions.

Starting .NET 4 they are marked as *CSE* — Corrupted State Exceptions.

To catch them you need to specify ***HandleProcessCorruptedStateExceptionsAttribute*** (doesn't work in .NET Core).

Only catch blocks with HPCSE are considered.

Only finally blocks in HPCSE methods are executed (and implicit finally for *using* construct).

This means that tons of **stack might not be unwound properly**.

CER also requires HPCSE.

StackOverflowException in .NET

You cannot catch it.

RuntimeHelpersEnsureSufficientExecutionStack() checks for the available stack:

- 512KB on x86, AnyCPU
- 2MB on x64
- 64/128KB on .NET Core

If not enough space is available, *InsufficientExecutionStackException* is thrown.

By default SOE kills the application. This can be configured in shim loading CLR to unload app domain in case of SOE.

How to catch SOE?

Catching with shim

Program.cs

```csharp
using System;

namespace StackOverflowGenerator
{
    class Program
    {
        public static void Main(string[] args)
        {
        }

        public static int Start(string args)
        {
            Console.WriteLine("Looping: " + args);
            Start(args);
            return 0;
        }
    }
}
```

Microsoft Visual Studio Debug Console

```
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
Looping: Argument
I'm done
```

StackOverflowCatcherViaShim.cpp

```cpp
using namespace mscorlib;
using namespace std;

LONG WINAPI VehExceptionHandler(_EXCEPTION_POINTERS* pExceptionInfo) {
    printf("%x\n", pExceptionInfo->ExceptionRecord->ExceptionCode);
    //0xE053534F - Soft SOE
    //0xE0434352 - CLR Exception
    //0xE06D7363 - C++ Exception
    if(pExceptionInfo->ExceptionRecord->ExceptionCode == 0xE053534F) {
        puts("SOE!");
    }

    return EXCEPTION_CONTINUE_SEARCH;
}

int main()
{
    HRESULT hr;
    ICLRMetaHost *pMetaHost = NULL;
    ICLRRuntimeInfo *pRuntimeInfo = NULL;
    ICLRRuntimeHost *pClrRuntimeHost = NULL;

    // build runtime
    hr = CLRCreateInstance(CLSID_CLRMetaHost, IID_PPV_ARGS(&pMetaHost));
    hr = pMetaHost->GetRuntime(L"v4.0.30319", IID_PPV_ARGS(&pRuntimeInfo));
    hr = pRuntimeInfo->GetInterface(CLSID_CLRRuntimeHost, IID_PPV_ARGS(&pClrRuntimeHost));

    // configure SOE handling
    ICLRControl *pCLRControl = NULL;
    pClrRuntimeHost->GetCLRControl(&pCLRControl);

    ICLRPolicyManager* pCLRPolicyManager = NULL;
    hr = pCLRControl->GetCLRManager(IID_ICLRPolicyManager, (PVOID*)&pCLRPolicyManager);
    hr = pCLRPolicyManager->SetActionOnFailure(FAIL_StackOverflow, eRudeUnloadAppDomain);

    // start runtime
```

Catching with shim

Program.cs

```
using System;

namespace StackOverflowGenerator
{
    class Program
    {
        public static void Main(string[] args)
        {
        }

        public static int Start(string args)
        {
            //Console.WriteLine("Looping: " + args);
            Start(args);
            return 0;
        }
    }
}
```

StackOverflowCatcherViaShim.cpp

```
using namespace mscorlib;
using namespace std;

LONG WINAPI VehExceptionHandler(_EXCEPTION_POINTERS* pExceptionInfo) {
    //printf("%x\n", pExceptionInfo->ExceptionRecord->ExceptionCode);
    //0xE053534F - Soft SOE
    //0xE0434352 - CLR Exception
    //0xE06D7363 - C++ Exception
    if(pExceptionInfo->ExceptionRecord->ExceptionCode == 0xE053534F) {
        //puts("SOE!");
    }

    return EXCEPTION_CONTINUE_SEARCH;
}

int main()
{
    HRESULT hr;
    LRMetaHost *pMetaHost = NULL;
    LRRuntimeInfo *pRuntimeInfo = NULL;
    LRRuntimeHost *pClrRuntimeHost = NULL;

    build runtime
    = CLRCreateInstance(CLSID_CLRMetaHost, IID_PPV_ARGS(&pMetaHost));
    = pMetaHost->GetRuntime(L"v4.0.30319", IID_PPV_ARGS(&pRuntimeInfo));
    = pRuntimeInfo->GetInterface(CLSID_CLRRuntimeHost, IID_PPV_ARGS(&pClrRuntimeHost));

    configure SOE handling
    LRControl *pCLRControl = NULL;
    lrRuntimeHost->GetCLRControl(&pCLRControl);

    LRPolicyManager* pCLRPolicyManager = NULL;
    hr = pCLRControl->GetCLRManager(IID_ICLRPolicyManager, (PVOID*)&pCLRPolicyManager);
    hr = pCLRPolicyManager->SetActionOnFailure(FAIL_StackOverflow, eRudeUnloadAppDomain);

    // start runtime
```

C:\Users\afish\Desktop\msp_windowsinternals\Debug\StackOverflowCatcherViaShim.exe

```
Process is terminated due to StackOverflowException.
```

StackOverflowCatcherViaShim.exe

StackOverflowCatcherViaShim.exe has stopped working

Windows can check online for a solution to the problem.

- Check online for a solution and close the program
- Close the program
- Debug the program

View problem details

Catching StackOverflowException in C#

1. Generate machine code on the fly.
2. Register VEH handler with P/Invoke.
3. Use „SetJump LongJump” like approach:
   1. Store registers.
   2. Call method generating SOE.
   3. Restore registers in VEH handler.
   4. Rely on VEH mechanism to perform the jump.
4. Continue.

```
56      // Exception info is passed in esp+8 which is fourth parameter according to fastcall calling convention
57      // Having signature correctly like (int, int, int, IntPtr) spils stack because of calling convention - .NET will remove 2 integers from it
58      // We need to fix ret 8 to ret 4 at the end
59      public static int CatchingVeh(int a, int b, int c, IntPtr pExceptionInfo)
60      {
61          //#define EXCEPTION_EXECUTE_HANDLER      1
62          //#define EXCEPTION_CONTINUE_SEARCH      0
63          //#define EXCEPTION_CONTINUE_EXECUTION (-1)
64
65          Console.WriteLine("Catching VEH!");
66
67          uint exceptionCode = (uint)Marshal.ReadInt32(Marshal.ReadIntPtr(pExceptionInfo));
68          Console.WriteLine("Exception code: " + exceptionCode.ToString("X"));
69          if(exceptionCode != 0xC00000FD)
70          {
71              // Not Stack Overflow Exception
72              // Return CONTINUE_SEARCH
73              return 0;
74          }
75
76          // We restore registers here using ExceptionInfo structure
77
78          Console.WriteLine("Info: " + pExceptionInfo.ToString("X"));
79          IntPtr pContextRecord = Marshal.ReadIntPtr(pExceptionInfo + 4);
80          Console.WriteLine("Context: " + pContextRecord.ToString("X"));
81
82          Marshal.WriteIntPtr(pContextRecord + 0xB8, (IntPtr)EipEspEbpEsiEdiEbxHolder[0]);
83          Marshal.WriteIntPtr(pContextRecord + 0xC4, (IntPtr)EipEspEbpEsiEdiEbxHolder[1]);
84          Marshal.WriteIntPtr(pContextRecord + 0xB4, (IntPtr)EipEspEbpEsiEdiEbxHolder[2]);
85          Marshal.WriteIntPtr(pContextRecord + 0xA0, (IntPtr)EipEspEbpEsiEdiEbxHolder[3]);
86          Marshal.WriteIntPtr(pContextRecord + 0x9C, (IntPtr)EipEspEbpEsiEdiEbxHolder[4]);
87          Marshal.WriteIntPtr(pContextRecord + 0xA4, (IntPtr)EipEspEbpEsiEdiEbxHolder[5]);
88
89          Console.WriteLine("Going back to Eip: " + EipEspEbpEsiEdiEbxHolder[0].ToString("X"));
90
91          // Return CONTINUE_EXECUTION
92          return -1;
93      }
94
95      // Method for registering VEH handler
96      [DllImport("Kernel32.dll", SetLastError = true)]
97      static extern int AddVectoredExceptionHandler(uint first, IntPtr addres);
```

C:\WINDOWS\system32\cmd.exe

```
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Preparing to throw
Catching VEH!
Exception code: C00000FD
Info: E02630
Context: E02B0C
Going back to Eip: 2FE4E98
I'm done
Press any key to continue . . .
```

Summary

Think about exceptions in async context in C#.

Understand what is your „last instruction” in case of *CONTINUE_EXECUTION*.

Be careful with returns in finally in SEH (and other technologies).

Don’t lose the details while rethrowing exceptions.

Automatically capture dumps on unhandled exceptions.

Know what you can’t handle.

Think about edge cases. What happens if your application crashes badly? How to survive if the state is corrupted? Can you revert it? Can yout at least make sure you don’t make it worse?

Have handlers for all unobserved exceptions. Consider notification for first chance exception.

Q&A

References

Jeffrey Richter - „CLR via C#”

Jeffrey Richter, Christophe Nasarre - „Windows via C/C++”

Mark Russinovich, David A. Solomon, Alex Ionescu - „Windows Internals”

Penny Orwick – „Developing drivers with the Microsoft Windows Driver Foundation”

Mario Hewardt, Daniel Pravat - „Advanced Windows Debugging”

Mario Hewardt - „Advanced .NET Debugging”

Steven Pratschner - „Customizing the Microsoft .NET Framework Common Language Runtime”

Serge Lidin - „Expert .NET 2.0 IL Assembler”

Joel Pobar, Ted Neward — „Shared Source CLI 2.0 Internals”

Adam Furmanek – „.NET Internals Cookbook”

https://github.com/dotnet/coreclr/blob/master/Documentation/botr/README.md — „Book of the Runtime”

https://blogs.msdn.microsoft.com/oldnewthing/ — Raymond Chen „The Old New Thing”

References

https://blog.adamfurmanek.pl/2016/04/23/custom-memory-allocation-in-c-part-1/ — allocating reference type on the stack

https://blog.adamfurmanek.pl/2018/03/24/generating-func-from-bunch-of-bytes-in-c/ — generating machine code in runtime

https://blog.adamfurmanek.pl/2018/04/07/handling-stack-overflow-exception-in-c-with-veh/ — handling SOE with VEH

https://blog.adamfurmanek.pl/2016/10/01/handling-and-rethrowing-exceptions-in-c/ — throwing exceptions and examining them with WinDBG

References

https://msdn.microsoft.com/en-us/magazine/dd419661.aspx?f=255&MSPPError=-2147217396#id0070035 — handling corrupted state exceptions

https://sellsbrothers.com/public/writing/askthewonk/HowdoestheThreadAbortExce.htm — *Thread.Abort()* internals

https://blogs.msdn.microsoft.com/oldnewthing/20100730-00/?p=13273/ — examining C++ exceptions in WinDBG

https://blogs.msdn.microsoft.com/oldnewthing/20160915-00/?p=94316 — the same revisited

https://marc.durdin.net/2011/12/windbg-and-delphi-exceptions-2/ — examinig Delphi exceptions in WinDBG

https://blog.quarkslab.com/visual-c-rtti-inspection.html — RTTI internals

http://www.openrce.org/articles/full_view/21 — SEH details

http://www.osronline.com/article.cfm?article=469 — SEH x64

https://reverseengineering.stackexchange.com/a/8419 — finding VEH handlers

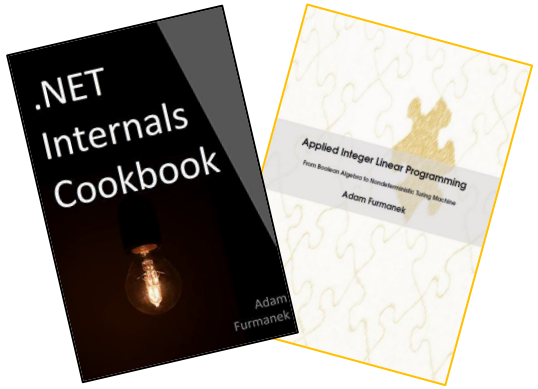
http://bytepointer.com/resources/pietrek_vectored_exception_handling.htm — VEH details

https://reverseengineering.stackexchange.com/a/14993 — VEH and VCH differences

https://web.archive.org/web/20150423173148/https://msdn.microsoft.com/en-us/magazine/cc163716.aspx — Reliability Features in .NET

https://translate.google.com/translate?hl=en&sl=auto&tl=en&u=http%3A%2F%2Fchepa.net%2Fall%2F2018%2F10%2F11%2Fособые-исключения-в-net-и-как-их-готовить%2F — Handling SOE in .NET with VEH (different way)

https://gynvael.coldwind.pl/?id=695 — DWARF in Linux



Random IT Utensils

IT, operating systems, maths, and more.

Thanks!

CONTACT@ADAMFURMANEK.PL

HTTP://BLOG.ADAMFURMANEK.PL

FURMANEKADAM