How do I find out more about the fail-fast exception that occurs when an exception in a PPL task goes unobserved?

devblogs.microsoft.com/oldnewthing/20230901-00

September 1, 2023

Raymond Chen

If an exception happens during the execution of a Parallel Patterns Library (PPL) `task`, the exception is stowed in the task, with the expectation that you will inspect the task to see whether or not it succeeded. But if you destruct the task without ever checking, then PPL fails fast with an unhandled exception, analogous with leaving an exception unhandled in non-task code.

```
__declspec(noinline)
~_ExceptionHolder()
{
    if (_M_exceptionObserved == 0)
    {
        // If you are trapped here, it means an exception thrown in task chain
        // didn't get handled. Please add task-based continuation to handle all
        // exceptions coming from tasks. this->_M_stackTrace keeps the creation
        // callstack of the task generates this exception.
        _REPORT_PPLTASK_UNOBSERVED_EXCEPTION();
    }
}
```

If your code is compiled in debug mode, PPL captures a stack trace at the point the first task in the task chain is created. If your code is compiled in release mode, PPL captures only the first stack in the trace.

When you get to the `_REPORT_PPLTASK_UNOBSERVED_EXCEPTION();`, you can look at the `this` pointer's `_M_stackTrace` to see where the task chain started.

In debug mode:

this	0x000001c9976ec0f0	Concurrency::details::_ExceptionHolder *

	_M_exceptionObserved	0	std::atomic<long>
	_M_stdException	{…}	std::exception_pt
	_M_stackTrace	{[capacity]=9	Concurrency::det
	[capacity]	9	void*
	[allocator]	allocator	std::_Compresse
	[0]	0x00007ff7d1b96792 {contoso.exe!StartSo…	void*
	[1]	0x00007ff7d1b96836 {contoso.exe!StudyS…	void*
	⋮	⋮	⋮

```
(8930.74bc): C++ EH exception - code e06d7363 (first chance)
Win32!Concurrency::details::_ExceptionHolder::~_ExceptionHolder+0x3e:
00007ff7`d1b93c5e cc              int     3
0:004> ?? this
struct Concurrency::details::_ExceptionHolder * 0x000001c9`976ec0f0
   +0x000 _M_exceptionObserved : std::atomic<long>
   +0x008 _M_stdException  : std::exception_ptr
   +0x018 _M_stackTrace    : Concurrency::details::_TaskCreationCallstack
0:004> ?? this->_M_stackTrace
class Concurrency::details::_TaskCreationCallstack
   +0x000 _M_SingleFrame   : 0x00007ff7`d1b96792 Void
   +0x008 _M_frames        : std::vector<void *,std::allocator<void *> >
0:004> ?? this->_M_stackTrace._M_frames
class std::vector<void *,std::allocator<void *> >
   +0x000 _Mypair          : std::_Compressed_pair<std::allocator<void
*>,std::_Vector_val<std::_Simple_types<void *> >,1>
0:004> ?? this->_M_stackTrace._M_frames._Mypair
class std::_Compressed_pair<std::allocator<void
*>,std::_Vector_val<std::_Simple_types<void *> >,1>
   +0x000 _Myval2          : std::_Vector_val<std::_Simple_types<void *> >
0:004> ?? this->_M_stackTrace._M_frames._Mypair._Myval2
class std::_Vector_val<std::_Simple_types<void *> >
   +0x000 _Myproxy         : 0x00000257`42a08840 std::_Container_proxy
   +0x008 _Myfirst         : 0x00000257`42a08000  -> 0x00007ff7`d1b96792 Void
   +0x010 _Mylast          : 0x00000257`42a08048  -> 0xabababab`fdfdfdfd Void
   +0x018 _Myend           : 0x00000257`42a08048  -> 0xabababab`fdfdfdfd Void
0:004> dps 0x00000257`42a08000 0x00000257`42a08048
00000257`42a08000  00007ff7`d1b96792 contoso!StartSomething+0x92 [something.cpp @
372]
00000257`42a08008  00007ff7`d1b96836 contoso!StudySomething+0x56 [something.cpp @
377]
...
```

Here we took advantage of what we learned earlier about compressed pairs and `std::vector`.

In release mode:

this	0x000001911e62aab0	Concurrency::details::_ExceptionHolder *	
	_M_exceptionObserved	0	std::atomic<long>
	_M_stdException	{…}	std::exception_pt
	_M_stackTrace	{[0]=0x00007ff61b9532f0 {contoso.exe!StartS…	Concurrency::det
	[0]	0x00007ff61b9532f0 {contoso.exe!StartSomet…	void*
	[Raw View]	{_M_SingleFrame=0x00007ff61b9532f0 {con…	Concurrency::det
	_M_SingleFrame	0x00007ff61b9532f0 {contoso.exe!StartSomet…	void*
	_M_frames	{ size=0 }	std::vector<void*>

```
(ae88.8188): C++ EH exception - code e06d7363 (first chance)
contoso!Concurrency::details::_ExceptionHolder::~_ExceptionHolder+0x15:
00007ff7`c4591be5 cc              int     3
0:004> dv
           this = 0x00000191`1e62aab0
0:004> ?? this
struct Concurrency::details::_ExceptionHolder * 0x00000191`1e62aab0
   +0x000 _M_exceptionObserved : std::atomic<long>
   +0x008 _M_stdException  : std::exception_ptr
   +0x018 _M_stackTrace    : Concurrency::details::_TaskCreationCallstack
0:004> ?? this->_M_stackTrace
class Concurrency::details::_TaskCreationCallstack
   +0x000 _M_SingleFrame   : 0x00007ff6`1b9532f0  Void
   +0x008 _M_frames        : std::vector<void *,std::allocator<void *> >

0:004> u 0x00007ff7`c45932f0 L1
contoso!StartSomething+0xe0: [something.cpp @ 377]
00007ff7`c45932f0 90              lea     r9,[rbp-68h]
```

We can see in the debugger that the frames vector is empty:

```
0:004> ?? this->_M_stackTrace._M_frames
class std::vector<void *,std::allocator<void *> >
   +0x000 _Mypair          : std::_Compressed_pair<std::allocator<void
*>,std::_Vector_val<std::_Simple_types<void *> >,1>
0:004> ?? this->_M_stackTrace._M_frames._Mypair
class std::_Compressed_pair<std::allocator<void
*>,std::_Vector_val<std::_Simple_types<void *> >,1>
   +0x000 _Myval2          : std::_Vector_val<std::_Simple_types<void *> >
0:004> ?? this->_M_stackTrace._M_frames._Mypair._Myval2
class std::_Vector_val<std::_Simple_types<void *> >
   +0x000 _Myfirst         : (null)
   +0x008 _Mylast          : (null)
   +0x010 _Myend           : (null)
```

Older versions of the Parallel Patterns Library captured only the top frame of the stack in a member variable named `_M_disassembleMe`:

this	0x000001ed8a3b6f10	Concurrency::details::_ExceptionHolder *	
	_M_exceptionObserved	0	volatile long
	_M_stdException	{…}	std::exception_pt
	_M_disassembleMe	0x00007ff61b9532f0 {contoso.exe!StartSomething(void), Line 377} }	void*

If C++/CX support is enabled, then there is also a member called `_M_winrtException`. Windows Runtime exceptions are stored in the `_M_winrtException` member; all others are stored in the `_M_stdException`.

You can then use previously-discussed techniques to extract the thrown exception and learn more about what went wrong.

```
0:004> ?? this->_M_stdException
class std::exception_ptr
   +0x000 _Data1           : 0x00000257`42a24880 Void
   +0x008 _Data2           : 0x00000257`42a24870 Void
0:004> .exr 0x00000257`42a24880
ExceptionAddress: 0000000000000000
   ExceptionCode: e06d7363 (C++ EH exception)
  ExceptionFlags: 00000001
NumberParameters: 4
   Parameter[0]: 0000000019930520
   Parameter[1]: 0000025742a24920
   Parameter[2]: 00007ff7bd414518
   Parameter[3]: 00007ff7bd310000
  pExceptionObject: 0000025742a24920
  _s_ThrowInfo    : 00007ff7bd414518
0:004> dps 0000025742a24920 L2
00000257`42a24920  00007ff7`bd4066b8 contoso!std::bad_alloc::`vftable'
00000257`42a24928  00007ff7`bd4066d0 contoso!`string'
```

Bonus reading: Improved exception reporting for C++ Windows Store Apps in Visual Studio 2013.