

StackWalk behaviour

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2006-06/msg00934.htm>

- *From:* spip_yeah@xxxxxxxxxx
 - *Date:* 21 Jun 2006 03:42:27 -0700
-

[Note: I am not sure if this belongs in programmer.kernel, or public.vc. I am clearly not writing kernel land code, but this newsgroup seems to have way more stalkwalk() related posts than any other]

I am adding an exception handler to certain of our build configurations in order to dump stack frame information for all threads in case of an unhandled exception.

I realize a debugger already provides that information. However, QA does not have access to any debuggers when testing the application, and knowing exactly where execution faulted is a huge help.

I have no problems dumping the stack frame from the thread that actually caused the fault. I install my handler (via `SetUnhandledExceptionFilter()`). When the handler catches an exception, I use the `CONTEXT` struct that I retrieve from the `EXCEPTION_POINTERS` struct to proceed to dumping the stack.

However, I then wanted to retrieve all threads for the application and actually dump the callstack for each thread. So I implemented that, and I am encountering the following issue: if I have N threads in my process, the information for the first thread that I try to `StackWalk()` results in no valid information. `StackWalk()` returns `TRUE` once, but I get a PC offset of 0.

The following threads are all fine. I typically had 3 threads, and only the second and third seemed to result in a valid callstack. If I had 2, then only the last would have a valid callstack written out.

I was initially running the "dump all threads" code from my exception handler. The callstack I printed out from the `EXCEPTION_POINTERS` struct would succeed. I then started iterating over my process thread and the 1st would fail, as I previously explained. It is worth noting that the thread that caused the fault would be correctly dumped out by the exception handler, at which point I attempt to dump the stack for all threads. The first thread that fails is actually the very same thread that caused the exception, and I was correctly able to dump the

StackWalk behaviour

callstack for that thread in the context of the exception.

I then tried to run this code independantly of the exception handler, so I now called it right after starting up my other threads, i.e. not in the context of an exception. I am seeing the same behaviour: the first fails (well, no functions ever return any errors).

Note that I have compiler optimizations disabled.

Someone with more win32 system knowledge might see something obvioulsy wrong I am doing, or might have encountered the same issue. Any idea what I am doing wrong?

This is what I do:

First, I get a snapshot:

```
HANDLE snapshot = CreateToolhelp32Snapshot( TH32CS_SNAPTHREAD, 0 );
```

I then iterate over each thread and only process the ones that belong to my process:

```
THREADENTRY32 thread;  
ZeroMemory( &thread, sizeof(thread) );  
thread.dwSize = sizeof(THREADENTRY32);  
  
DWORD processId = GetCurrentProcessId();  
DWORD threadId = GetCurrentThreadId();  
  
BOOL found = Thread32First( snapshot, &thread );  
while(found)  
{  
if( thread.th32OwnerProcessID == processId &&  
thread.th32ThreadID != threadId )  
{  
CONTEXT ctx;  
HANDLE hThread;  
  
hThread = OpenThread( THREAD_QUERY_INFORMATION |  
THREAD_GET_CONTEXT, 0, thread.th32ThreadID );  
if( hThread == NULL ) // crude error handling, just  
notifying  
PrintDbg( "Unable to OpenThread()\n" );  
  
if( !GetThreadContext( hThread, &ctx ) ) // crude error  
handling, just notifying  
PrintDbg( "Unalbe to get thread context\n" );  
  
DumpCallStack( &ctx, hThread );  
  
CloseHandle( hThread );
```

StackWalk behaviour

StackWalk behaviour

```
}  
  
found = Thread32Next( snapshot, &thread );  
}
```

Thank you.

.