MSVC run-time bug in Malloc in Debug Mode

<table>
<thead>
<tr>
<th>Author</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chris Mullin</td>
<td>MSVC run-time bug in Malloc in Debug Mode</td>
</tr>
<tr>
<td>#1 / 7</td>
<td>Fastest VPN for China</td>
</tr>
<tr>
<td></td>
<td>Unblock any Site. Try it Risk Free. 256-Bit SSL. High Speed Guaranteed!</td>
</tr>
</tbody>
</table>

I've discovered a very {*filter*} bug in Malloc when building and running in debug mode. I'm hoping someone here has come across this and knows a good workaround.

(This is crossposted to microsoft.public.vc.stl because the problem originally manifested itself during repeated copies of a large STL map, which use "new" which uses "malloc")

I'm using MSVC 6, service pack 3.

The crux of the problem is that the 4 billionth malloc that I do fails - in fact it causes a hardcoded breakpoint (an int 3) inside DBGHeap.C. This, of course, takes down the production systems about every 11 days (we leave everything running in Debug mode so that we can debug via Dr. Watson Dump files if we need to).

The culprit is the comparison on line 337 in DBGHeap.c in the standard C run-time library:

```c
/* break into de{*filter*} at specific memory allocation */
if (iRequest == _crtBreakAlloc)
  _CrtDbgBreak();
```
lRequest is a static variable that is incremented each time a malloc or realloc is done. _crtBreakAlloc is a debug variable with a default value of -1.

On the 4 billionth malloc, when lRequest is equal to 0xFFFFFFFF, the comparison is true and a software breakpoint is called.

Note that the value that I print out in the following program, "MallocCounter", is, in reality, a static (private) variable inside DBGHeap.c - the next variable defined in that C file is not static, so I can grab its address, go back the size of a long, and get to the variable in question.

From DBGHeap.c:

```c
static long _lRequestCurr = 1;      /* Current request number */
umbnails/CRTIMP long _crtBreakAlloc = -1L;          /* Break on allocation by request number */
```

The value _lRequestCurr is the one that I’m going after...

Here is a sample program that highlights the problem - it doesn’t require any special settings at all. To hurry things up, I set the Malloc counter to a high value to start with, but it does happen naturally on a PIII 400 in about 6 hours.

```c
#include <stdio.h>
#include <stdlib.h>
extern "C"
{
    extern long _crtBreakAlloc;

    int main()
    {
        int *x;
        long *MallocCounter = &_crtBreakAlloc - 1;
        printf("BrakAlloc: %d\n", _crtBreakAlloc);
        printf("MallocCounter: %d\n", *MallocCounter);
        *MallocCounter = 0xFFEFFFF0;
        while(1)
        {
            x = (int*) malloc( sizeof( int ) );
            if ( ( *MallocCounter & 0xFFFF ) == 0)
            {
                printf("Counter [%X]\n", *MallocCounter);
            }
            free(x);
        }
        return 0;
    }
}
```
MSVC run-time bug in Malloc in Debug Mode

_crtBreakAlloc and the hard-coded break point are there to help debug memory leaks. One person's bug is another person's feature.... In any case, if the only reason you are running a debug build is for Dr. Watson dumps, why not just build your release builds with debug information? (You may also want to disable optimizations, if ease of debugging is important.) That way you would still be able to debug the dumps, but you would be using the release version of the C run-time, hence bypassing the debug heap manager and this bug/feature.
If you absolutely must use the debug heap manager, you'll probably need to write your own operator new which ensures that _crtBreakAlloc is not the same as the current memory allocation, probably using _CrtIsMemoryBlock() and _CrtSetBreakAlloc().

--Carter

Quote:

> I've discovered a very {*filter*} bug in Malloc when building and running in debug mode. I'm hoping someone here has come across this and knows a good workaround.
> (This is crossposted to microsoft.public.vc.stl because the problem originally manifested itself during repeated copies of a large STL map,
> which use "new" which uses "malloc" )
> I'm using MSVC 6, service pack 3.
> The crux of the problem is that the 4 billionth malloc that I do fails - in fact it causes a hardcoded breakpoint (an int 3) inside DBGHeap.C. This,
of course, takes down the production systems about every 11 days (we leave everything running in Debug mode so that we can debug via Dr. Watson Dump files if we need to).
The culprit is the comparison on line 337 in DBGHeap.c in the standard C run-time library:
/* break into de{*filter*} at specific memory allocation */
if (lRequest == _crtBreakAlloc)
    _CrtDbgBreak();
lRequest is a static variable that is incremented each time a malloc or realloc is done. _crtBreakAlloc is a debug variable with a default value of -1.
On the 4 billionth malloc, when lRequest is equal to 0xFFFFFFFF, the comparison is true and a software breakpoint is called.
Note that the value that I print out in the following program,
"MallocCounter", is, in reality, a static (private) variable inside DBGHeap.c - the next variable defined in that C file is not static, so I can grab it's address, go back the size of a long, and get to the variable in question.
From DBGHeap.c:
static long _lRequestCurr = 1;    /* Current request number */
_CRTIMP long _crtBreakAlloc = -1L;    /* Break on allocation by request number */
The value _lRequestCurr is the one that I'm going after...
Here is a sample program that highlights the problem - it doesn't require any special settings at all. To hurry things up, I set the Malloc counter to a high value to start with, but it does happen naturally on a PIII 400 in about 6 hours.
#include <stdio.h>
#include <stdlib.h>
extern "C"
{
    extern long _crtBreakAlloc;
}
int main()
{
    int *x;
    long *MallocCounter = &_crtBreakAlloc - 1;
    printf("BrakAlloc: %d\n", _crtBreakAlloc);
    printf("MallocCounter: %d\n", *MallocCounter);
    *MallocCounter = 0xFFEFFFF0;
    while(1)
    {
        x = (int*) malloc( sizeof( int ) );
        if ( (*MallocCounter & 0xFFFF ) == 0)
>         {               
>             printf("Counter [%X]\n", *MallocCounter);
>         }               
>         free(x);      
>     }              
>     return 0;       
> }                  
> --                
> Chris Mullins

Quote:
> The crux of the problem is that the 4 billionth malloc that I do fails - in fact it causes a hardcoded breakpoint (an int 3) inside DBGHeap.C. This, of course, takes down the production systems about every 11 days (we leave everything running in Debug mode so that we can debug via Dr. Watson Dump files if we need to).

The Debug version of the CRT applies a unique tag to each memory allocation. In this case, I would guess that the counter that it is using is a 32-bit integer that is being overflowed.  
1) Use the Release version of the CRT in your Debug builds -- this may leave enough info for Dr Watson.  
2) You may be able to disable this feature with
CrtSetDbgFlag( _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG) & ~_CRTDBG_ALLOC_MEM_DF );
3) Periodically re-set the counter using code similar to that in your example. I know it's a hack, but there's a good chance it'll work...may have strange interactions if you are using any of the other _CrtXXXX functions.
Michael Christie

Joe O'Lear #4 / 7

MSVC run-time bug in Malloc in Debug Mode

It's possible to replace the debug malloc with your own version. That's what most custom heap managers (such as Hoard) do. However if that's not an option, you might consider some kind of SEH frame that checks for EXCEPTION_DEBUG_BREAKPOINT and possibly returns an RETRY code (though I don't know how feasible that would be).
Still, (and this is probably the kind of tangential response you don't want to hear, but...) a production system is *not* made to run on DEBUG binaries. I wouldn't be surprised if you got that type of response from Microsoft (assuming you ever reach them). There are alternatives. Generate debug info for your release binaries too. Instead of relying on Dr Watson dump files, set up some SEH frames possibly an SEH translator if you use C++ exceptions. Within your SEH handler, make use of the symbolic debugging functions (the SymXXXX functions from IMAGEHLP.DLL or DBGHELP.DLL) from within the translator. You can walk the entire stack and print out a nice stack trace of every method leading right up to the one that caused the exception. This is precisely what we do in our product. Our logs come back with full stack traces just like while Debugging in Visual Studio. You can read all about how to use these functions, plus get some nice wrappers from John Robbins' "Debugging Applications". Very good book.

Joe

Quote:
> I've discovered a very {*filter*} bug in Malloc when building and running in
debug mode. I'm hoping someone here has come across this and knows a good workaround. (This is crossposted to microsoft.public.vc.stl because the problem originally manifested itself during repeated copies of a large STL map, which use "new" which uses "malloc".)

I'm using MSVC 6, service pack 3. The crux of the problem is that the 4 billionth malloc that I do fails - in fact it causes a hardcoded breakpoint (an int 3) inside DBGHeap.C. This, of course, takes down the production systems about every 11 days (we leave everything running in Debug mode so that we can debug via Dr. Watson Dump files if we need to).

Sun, 05 Jan 2003 03:00:00 GMT

Chris Mullin #5 / 7

MSVC run-time bug in Malloc in Debug Mode

Quote:

> The Debug version of the CRT applies a unique tag to each memory allocation. In this case, I would guess that the counter that it is using is a 32-bit integer that is being overflowed.

I can say with absolute certainty that you are correct.

Quote:

> 1) Use the Release version of the CRT in your Debug builds -- this may leave enough info for Dr Watson.

I'll give this a try.

Quote:

> 2) You may be able to disable this feature with CrtSetDbgFlag(_CrtSetDbgFlag(_CRTDBG_REPORT_FLAG) & ~_CRTDBG_ALLOC_MEM_DF);
Unfortunately after looking at the assembly code, nothing like that's going to work. Ah well, it was a good idea.

Quote:
> 3) Periodically re-set the counter using code similar to that in your example. I know it's a hack, but there's a good chance it'll work...
> may have strange interactions if you are using any of the other _CrtXXXXX functions.

I think that's the short-term fix, while I spend a few days looking into the implications of re-releasing our production servers build in Release mode (with Debug Info on, and probably most optimizations turned off).
--
Chris Mullins

Sun, 05 Jan 2003 03:00:00 GMT

Gary Nebbet 
#6 / 7

MSVC run-time bug in Malloc in Debug Mode

Hello Chris,
Something else that you could try is to use the UnhandledExceptionFilter to dismiss all breakpoint exceptions. The success of this method would depend upon frame-based exception handlers ignoring (e.g. return EXCEPTION_CONTINUE_SEARCH) EXCEPTION_BREAKPOINT exceptions. You could even use your technique for finding the value of _lRequestCurr and only dismiss the exception when _lRequestCurr has a particular value.
Here is some sample code:
#include <windows.h>
#include <stdio.h>
LONG WINAPI DismissBreakPoint(PEXCEPTION_POINTERS ep)
{
    return ep->ExceptionRecord->ExceptionCode == EXCEPTION_BREAKPOINT
        ? ep->ContextRecord->Eip++, EXCEPTION_CONTINUE_EXECUTION :
        EXCEPTION_CONTINUE_SEARCH;
}

int main()
{
    SetUnhandledExceptionFilter(DismissBreakPoint);
    printf("1\n");
    DebugBreak();
printf("2\n");
volatile int *x = 0, y = *x;
printf("3\n");
return 0;

Quote:
}

Gary

Mon, 06 Jan 2003 03:00:00 GMT

Chris Mullin  
#7 / 7

MSVC run-time bug in Malloc in Debug Mode

Quote:

> >This, of
> >course, takes down the production systems about
every 11 days (we leave
> >everything running in Debug mode so that we can
debug via Dr. Watson Dump
> >files if we need to).
> I create an "NDebug" configuration. No optimizations.
Full debug info
left
> in the builds. Debug defines off. Release libraries. I
prefer this

setup.
I looked at this as well, but it does have a few drawbacks.
When I examine Dump files, if I see "0CCCCCCCCC" or
"0CDCDCDCD" in a
register, then I immediatly have a good lead as to the cause
of a crash.
There are several other features as well that I like, that are
only
available with the Debug CRT libraries.
I think the answer that I'm going to go with is to remove the
"_int 3" from
the call that's given me a hard time. Either by rebuilding the
libraries, or
by simply using a binary editor (replace the int3 in question
with a no-op)
on the CRT Library on our build machine.
--
Chris

Tue, 07 Jan 2003 03:00:00 GMT
1. MSVC run-time bug in Malloc in Debug Mode
2. Run time Release mode error but no Debug mode error
3. MSVC 7.0 compiler bug in debug mode
4. MSVC Runtime error with Just In Time debugging
5. Program runs fine in debug mode, but crashes in release mode
6. Design-time vs Run-time mode
7. Error in Debug mode using MSVC
8. Serious bug in MSVC runtime library
9. Q: malloc and type info - fatal run-time error
10. malloc fails sooner when using multithreaded DLL runtime
11. Use debug version of malloc and free cause a puzzled BUG
12. URGENT: Debug mode vs. Real Time