Freeing Memory Allocated by the HDF5 Library

Several functions in the HDF5 C API return buffers allocated by the HDF5 Library. When application code uses a different library for memory management than the HDF Library, a corrupt heap or a resource leak can occur when these allocated buffers are freed. This is most commonly a problem on Windows systems since Microsoft implements C library functions in Visual Studio---specific libraries which do not share heap state.

This document describes this problem and the steps users can take to mitigate the problem. This document also introduces the new `H5free_memory()` function.

Introduced with

HDF5 Release 1.8.13

May 15, 2014

For more information about The HDF Group, see www.hdfgroup.org.

Contents

1. Introduction 4

2. The Windows C Run-time (CRT) 5

3. Potentially Affected API Calls 6
   3.1. Functions that Return a Pointer 6
   3.2. Functions that Have a ** OUT Parameter 6
   3.3. Other 6

4. Mitigation 7
   4.1. Use the Same Memory Manager/Correct C Run-time Everywhere 7
   4.2. Use the H5free_memory() Function 7

1. Introduction

In the HDF5 Library, responsibility for the allocation and freeing of memory is usually the responsibility of the same component: either the library or the user's code. When data that would normally be stored in dynamically-allocated memory must be returned from the library, the user is usually asked to allocate a buffer which is passed to the function and then filled by the library. The complication is that the user must be able to determine the buffer's size. The mechanism for this is for the user to make a preliminary call, passing a NULL pointer in for the buffer. The function will then return the appropriate number of bytes for the user to allocate. See the example below.

```
ssize_t     size;
size_t      bufsize;
hid_t       object_id;
char        *comment;
…
size = H5Oget_comment(object_id, NULL, bufsize); /* determine size */
bufsize = size;
comment = (char *)malloc(bufsize * sizeof(char));
size = H5Oget_comment(object_id, comment, bufsize);   /* fill buffer */
```

Example 1. Determining the buffer size with a preliminary call

There are, however, several API calls in which the buffer is allocated by the HDF5 Library and returned to the user who is responsible for freeing it. This can be a problem when memory is managed via different libraries as it can result in resource leaks or a corrupted heap. This heap corruption can result in subtle bugs that can be very difficult to reproduce and diagnose. In most cases, having the library allocate memory and the application free it is not a problem since memory operations will resolve down to the operating system's memory manager; however, there are cases where this is not true. For example, a debug memory manager may be in use by the application code but not the library. A complication that is unique to Windows is that the C standard library functions are implemented in Visual-Studio-specific shared libraries. When different versions of Visual Studio are used to compile the library and user code, the allocate and free calls are made in different libraries, which do not share state, leading to the previously mentioned resource and corruption issues.

2. The Windows C Run-time (CRT)

Microsoft implements the standard C library functions in debug and release libraries that are specific to each version of Visual Studio[1]. Each library is a separate entity and maintains its own internal CRT object state, file handles, and heap information. Creating an object in one CRT and destroying it in another CRT may appear to work but can cause corruption of one CRT and resource leaks in the other.

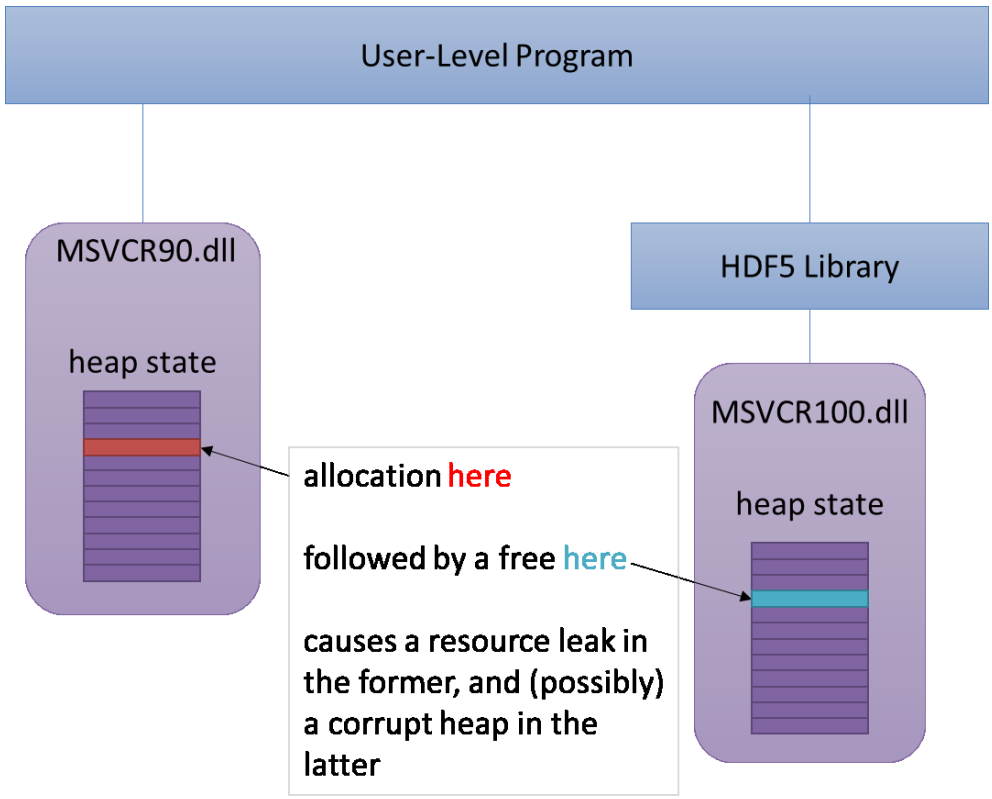

Figure 1. How the heap corruption and resource leaks might occur

These problems are normally avoided on Windows by ensuring that all components that can return CRT resources are linked to the same CRT dynamic link library (DLL). Unfortunately, even debug and release CRTs are housed in separate DLLs, so this is not an easy solution to implement. Using static linkage does not avoid this problem since separate copies of the CRT are created in each statically linked component.

[1] The names of these libraries are of the form MSVCR<#>.dll, where <#> is the Visual Studio version. For example, MSVCR110.dll corresponds to Visual Studio 11.0 (2012).

3. Potentially Affected API Calls

This is a list of all API calls that are potentially affected. Note that these have not all been validated as having allocate/free issues.

3.1. Functions that Return a Pointer

- `H5Iobject_verify`
- `H5Iremove_verify`
- `H5Isearch`
- `H5Pget_class_name`
- `H5Pget_driver_issue`
- `H5Tget_member_name`
- `H5Tget_tag`

3.2. Functions that Have a ** OUT Parameter

- `H5Eget_auto1`
- `H5Eget_auto2`
- `H5Fget_vfd_handle`
- `H5Lunpack_elink_val`
- `H5Pget_buffer`
- `H5Pget_elink_cb`
- `H5Pget_fapl_multi`
- `H5Pget_file_image`
- `H5Pget_mdct_search_cb`
- `H5Pget_type_conv_cb`
- `H5Pget_vlen_mem_manager`

3.3. Other

- `H5Fget_mdc_config` (trace_file_name struct member)
- `H5Pget_mdc_config` (trace_file_name struct member)

4. Mitigation

There are several potential solutions to the problem of freeing memory allocated by the HDF5 Library.

4.1. Use the Same Memory Manager/Correct C Run-time Everywhere

Both application code and the HDF5 Library must use the same memory allocator. When using Visual Studio, both the Visual Studio version and release/debug state must be identical. As of HDF5 1.8.12, this is the only available solution.

4.2. Use the H5free_memory() Function

A new function called `H5free_memory` has been created and is essentially a thin wrapper for the run-time's free() call. This function would be used to free any memory allocated by the library. This solution has the advantages of being extremely easy to implement and intuitive to use. It can also be used as a solution with legacy API calls, so it would be necessary even if we modify the HDF5 API. This function will also be extremely useful when HDF5 is wrapped for use with managed languages such as Java, .NET, and Python so that the wrappers can properly clean up resources.

See the `H5free_memory` entry in the *HDF5 C Reference Manual* for more information.

Note that the creation of this function does not imply that it will be acceptable for new API calls to be created that return library-allocated memory. The preferred mechanism will still be to use the "preliminary call" scheme described in the "Introduction" on page 4 where the user allocates the buffer.