RFC: Allocate/Free Mismatches in HDF5 Filter Code on Windows

Dana Robinson

As data pass through the filter chain on their way from the dataset to the user, the buffers in which the data are stored may need to be re-allocated. This becomes a problem when third-party filters link to a different memory manager than the HDF5 library. In this situation, the allocating library will slowly leak memory and the freeing library will corrupt its heap, exhibit failures, etc.

On POSIX systems, this is usually not a problem since the C library is a part of the operating system. An important exception is when debug or high-performance memory managers are being used. On Windows, however, this is often a problem since Microsoft implements C library functions in a collection of Visual-Studio- and configuration-specific libraries, which do not share heap state.

This RFC describes this problem and steps the user can take to mitigate the problem. It also introduces the new `H5(re)allocate_memory()` functions, which will appear in HDF5 1.8.15 (release date: May 2014).

Introduction

The Underlying Problem

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.

Example:

```
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 */
```

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.

In an effort to mitigate this problem, a new H5free_memory() call was added to the C API in HDF5 1.8.13 which exposes the library's free() call. This function can be used to ensure that memory allocated by the library is freed by the library. Unfortunately, there is still a situation where this allocate/free mismatch can occur.

Memory Reallocation In Third-Party Filters

From the H5Zregister() API documentation:

> *"The filter should perform the transformation in place if possible. If the transformation cannot be done in place, then the filter should allocate a new buffer with malloc() and assign it to *buf, assigning the allocated size of that buffer to *buf_size. The old buffer should be freed by calling free()."*

The problem with these instructions is that, if the filter is linked to a different memory allocation library (or C run-time), then the memory allocated by the filter's library will be freed in the HDF5 library, and vice-versa. This is illustrated in the figure below.

The figure shows the allocation of a buffer in the library, into which data from storage are written. This buffer comes from the HDF5 library's allocator (the Windows C run-time is used here). The buffer is then passed to the filter, which may need to enlarge it since the processed data may be larger than the raw data. This is shown in the center region as a free followed by an allocation. In this case, the filter is linked to an independent memory allocator, which obviously does not track the original buffer in its heap structures. Depending on the allocator, the free here could result in a memory leak, an immediate crash, or heap corruption. If the program continues, the library will experience the same problem when it attempts to free the buffer allocated by the filter.

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. Each library is a separate entity and maintains its own internal CRT object state, including 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.

These problems are normally avoided on Windows by ensuring that all components that can return CRT resources are linked to the same CRT 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.

Note that it can be extraordinarily difficult to ensure that memory is allocated and freed in the same library on Windows. Not only are there different dlls for shared vs. static and the various versions of Visual Studio, but you also have to deal with compatibility libraries (Visual Studio 2012 can use a special version of the C run-time that is compatible with Windows XP, for example) and service pack libraries. These "dll variants" will all have the exact same name, making them difficult to tell apart. There are also potential path issues, where it looks like a particular dll is being used, but an incorrect plugin search path causes an incorrect library to be loaded.

Mitigation

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

Use the Same Memory Manager/Correct C Run-Time Everywhere

Both filter 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 1.8.14, this is the only available solution.

As noted previously, this can be very difficult to pull off in practice. It can also be extremely difficult to debug problems when things go awry.

Create an H5(re)allocate_memory() and Functions

Like H5free_memory(), these functions would be thin wrappers around the HDF5 C library's HDmalloc() and HDrealloc() functions. Optionally, a Boolean parameter passed to H5allocate_memory() could indicate whether the memory should be cleared or not.

```
herr_t H5allocate_memory(size_t size, hbool_t clear, /*out*/ void **buf)
herr_t H5reallocate_memory(size_t size, /*in-out*/ void **buf)
```

As a historical note, the odd naming scheme (H5free_memory, etc.) is due to the "H5free()" not being entirely clear as to what was being freed. This would be particularly dangerous since it takes a void pointer, allowing any buffer to be passed in without the compiler complaining.

These functions would be used to allocate memory that will later be freed by the library. Their only intended use at this time is by filter authors. The memory management schemes used by other API calls will remain unchanged. This solution has the advantages of being extremely easy to implement and intuitive to use.

Tentative reference manual pages for these functions are available as an appendix to this RFC.

Add Functions that Set/Get Memory Management Functions

These functions would probably be constructed along the lines of H5Pset_vlen_manager() and could either be filter-pipeline-specific H5Z* functions or could be more generic H5* functions that apply to the library as a whole, supplanting even the HDmalloc(), etc. functions.

Testing

Ideally, the best way to test this functionality would be to locate an open-source, platform-independent malloc/free replacement (perhaps Dmalloc) and link that to a purpose-built testing filter and plugin that always reallocates the input memory buffer.

An additional, one-time, test will be to ensure that the c-blosc filter and plugin work when its configuration does not match the HDF5 library's.

We may also want to use this replacement allocator to write a small test that exercises the memory-returning functions that must use H5free_memory().

Recommendations

The easiest, least painful solution is to add H5(re)allocate_memory() functions to the API, starting with HDF5 1.8.15 (release date: May 2015). This solution is the most straightforward solution to the "multiple CRT dlls" problem and will be easy for maintainers to implement. It's also the easiest to use from managed languages since it's a simple API call and does not involve passing function pointers.

Allowing memory management functions to be specified also holds a certain appeal, however this would be a more complicated solution to implement. Given the need to get HDF5 1.10 out the door next year, it's probably better to opt for the more expedient solution and save a more generalized memory management scheme for HDF5 1.12.

Additionally, the H5Z section of the reference manual will need to be updated to indicate that filter implementers should use the new functions when they have to re-allocate a buffer.

We should also proactively contact the filter authors listed on our website to inform them of this problem so they can fix their code.

Suggested Task List

- Add H5allocate_memory() and H5reallocate_memory() functions to HDF5 C library.
- Evaluate a malloc() replacement for testing purposes.
- Modify the malloc() replacement for inclusion in the library. This is likely to require writing CMakeLists.txt files for the library, storing the source in our version control system, etc.
- Create a test filter and plugin that use the replacement memory allocation library.
- Create a test that exercises the filter and plugin to ensure that they do not corrupt the heap.
- Update the reference manual and user's guide with new best practices.
- Contact known filter authors and suggest that they modify their code.
- (Optional) Create a test, linked to the replacement allocator, that ensures that H5free_memory correctly frees memory without corrupting the heap.
- (Optional) Investigate unifying our memory management functions/schemes for a future release.

Acknowledgements

This work is being internally funded by The HDF Group.

Revision History

December 1, 2014: Version 1 circulated for comment among selected members of The HDF Group.

References

1. The HDF Group. "HDF5 Documentation", http://www.hdfgroup.org/HDF5/doc/doc-info.html (December 1, 2014).
2. Microsoft MDSN. "Potential Errors Passing CRT Objects Across DLL Boundaries (Visual Studio 2012)", http://msdn.microsoft.com/en-us/library/ms235460%28v=vs.110%29.aspx (December 1, 2014).

3. Microsoft MDSN. "C Run-Time Libraries (Visual Studio 2012)", http://msdn.microsoft.com/en-us/library/abx4dbyh%28v=vs.110%29.aspx (December 1, 2014).
4. Microsoft MDSN. "How to link with the correct C Run-Time (CRT) library", http://support.microsoft.com/kb/140584 (December 1, 2014).
5. The Dmalloc website. http://dmalloc.com/(December 1, 2014).

1) Appendix: Proposed H5allocate_memory() Reference Manual Page

Name: H5allocate_memory

Signature:

```
herr_t H5allocate_memory(size_t size, hbool_t clear,
    /*OUT*/ void **buf)
```

Purpose:

Allocates memory that will later be freed internally by the HDF5 library.

Description:

In order to avoid heap corruption, allocated memory should be freed using the same library that initially allocated it. In most cases, the HDF5 API uses resources that are allocated and freed entirely by either the user or the library so this is not a problem. In rare cases, however, HDF5 API calls will free memory that the user allocated. This function allows the user to safely allocate this memory.

At this time, the only intended use for this function is for allocating memory that will be returned to the library (and eventually the user) as a data buffer from a third-party filter.

Note:

This function is intended to have the semantics of malloc() and calloc().

It is particularly important to use this function to allocate memory on Windows. The C standard library is implemented in dynamic link libraries (dlls) known as the C run-time (CRT). Each version of Visual Studio comes with multiple versions of the CRT dlls (debug, release, etc.) and allocating and freeing memory across dll boundaries can cause resource leaks and subtle bugs due to heap corruption.

Only use this function to allocate memory inside third-party HDF5 filters. It will generally not be safe to use this function to allocate memory for any other purpose.

Even when using this function, it is still best to ensure that all components of a C application are built with the same version of Visual Studio and configuration (Debug or Release) and thus linked against the same CRT.

Parameters:

`size_t size`	Size of the buffer that will be allocated.
`hbool_t clear`	Whether or not the new buffer should be memset to zero.
`void **mem`	OUT: Pointer to a buffer that will be allocated.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

The mem parameter will be set to NULL on failure.

2) Appendix: Proposed H5allocate_memory() Reference Manual Page

Name: H5reallocate_memory

Signature:

```
herr_t H5reallocate_memory(size_t size,
     /*IN-OUT*/ void **buf)
```

Purpose:

Resizes (possibly reallocating) memory that will later be freed internally by the HDF5 library.

Description:

In order to avoid heap corruption, allocated memory should be freed using the same library that initially allocated it. In most cases, the HDF5 API uses resources that are allocated and freed entirely by either the user or the library so this is not a problem. In rare cases, however, HDF5 API calls will free memory that the user allocated. This function allows the user to safely resize this memory.

At this time, the only intended use for this function is for resizing memory that will be returned to the library (and eventually the user) as a data buffer from a third-party filter.

Note:

This function is intended to have the semantics of realloc().

If the input buffer is NULL, the function works like H5allocate_memory() with a clear parameter of FALSE.

The input buffer MUST either be NULL or have been allocated by H5allocate_memory() since the input buffer may be freed by the library.

It is particularly important to use this function to resize memory on Windows. The C standard library is implemented in dynamic link libraries (dlls) known as the C run-time (CRT). Each version of Visual Studio comes with multiple versions of the CRT dlls (debug, release, etc.) and allocating and freeing memory across dll boundaries can cause resource leaks and subtle bugs due to heap corruption.

Only use this function to resize memory inside third-party HDF5 filters. It will generally not be safe to use this function to resize memory for any other purpose.

Even when using this function, it is still best to ensure that all components of a C application are built with the same version of Visual Studio and configuration (Debug or Release) and thus linked against the same CRT.

Parameters:

`size_t size`	Size of the buffer that will be allocated.
`void **mem`	IN-OUT: Pointer to a buffer that will be allocated.

May be NULL.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

The mem parameter will be left unchanged on failure.