Diskheap Library

Version 1.5.1
March 2002

Gray Watson

Diskheap Library

The *Diskheap* library provides functionality similar to the memory functions `malloc`, `realloc`, `free`, etc. but on disk. With the library you can write simple data structure storage wrappers for hash tables, trees, virtual file systems, etc.. The library should be reasonably portable to most Unix systems. Please provide feedback to the author if you have problems with it.

The package includes the library, configuration scripts, test program, and documentation. Online documentation as well as the full source is available at URL `http://256.com/sources/diskheap/`.

My contact information is available on the web page. I can be reached with any questions or feedback. Please include the version number of the library that you are using and your machine and operating system types.

Gray Watson.

1 Library Copying and Licensing Conditions

2 Installation Notes, Defintions, and Sample Code

2.1 How to Install the Library

To configure, compile, and install the library, follow these steps carefully.

1. Make sure you have the latest version of the library available from the home page `http://256.com/sources/diskheap/`.
2. Type *`sh ./configure`* to configure the library. You may want to first examine the '`config.help`' file for some information about configure. *`sh ./configure --help`* lists the available options to configure. Configure should generate the '`Makefile`' and configuration files automatically.
3. You may want to examine the '`Makefile`' and '`conf.h`' files created by configure to make sure it did its job correctly.
4. Typing *`make`* should be enough to build the '`libdiskheap.a`' library. If it does not work and you figure your problem out, please send me some notes so future users can profit from your experiences.
5. Typing *`make light`* should build and run the '`diskheap_t`' test program through a set of light trials. By default this will execute '`diskheap_t`' 5 times – each time will execute 1000 Diskheap operations in a very random manner. Anal folks can type *`make heavy`* to up the ante. Use *`diskheap_t --usage`* for the list of all '`diskheap_t`' options.
6. Typing *`make install`* should install the '`libdiskheap.a`' library in '`/usr/local/lib`' and the '`diskheap.h`' include file in '`/usr/local/include`'. You may have specified a '`--prefix=PATH`' option to configure in which case '`/usr/local`' will have been replaced with '`PATH`'.

2.2 General Memory Terms and Concepts

To programmers, a heap is a bunch (or pile) of memory. Programs can make calls to allocate some memory from the heap to process a file (for example). When the program is done with the memory, it can free it back to the heap so that other parts of the program can use it. Heap memory is most useful when you do not know ahead of time the memory necessary to complete a task. The file could be large or small and allocating a small static space wouldn't be enough to process a large file while allocating a large space might waste system resources.

This ability to dynamically allocate space so you can perform a task or store a value is called (drum roll please) dynamic memory. Dynamic memory functions such as `malloc`, `realloc`, `free`, etc. provide dynamic storage functions for memory inside your application. The Diskheap library provides dynamic storage functionality similar to the in-memory heap functions but on disk.

When some space is allocated in the heap, the library returns its location as a block-number and offset pair of 32-bit unsigned integers. Both the block-number and the offset

must be provided to retrieval, update, delete, and other functions to reference this space in the future.

2.3 Small Sample of Code Showing Usage of the Library

Below is a simple example of what you can do with the library. *Please note* that although it gives you some idea about the basic functionality of the library it is not really doing something useful.

```
main()
{
        diskheap_t *diskheap_p;
        unsigned int block_n, offset, size;
        char *str_p;
        int ret;

        /* create a new diskheap file called 'heap' */
        diskheap_p = diskheap_create("heap", 0, 0, 0, 0, 0L);

        /*
         * Store the string 'hello there' (size 12 bytes) in the heap.
         * The variables block_n and offset get set with the location
         * of the string.
         */
        ret = diskheap_store(diskheap_p, "hello there", 12, 0, &block_n,
                             &offset);
        /* ret should be checked against DISKHEAP_ERROR_NONE */

        /*
         * Update the 'hello there' string and replace with 'hello
         * there again'.  You pass in the block_n and offset variables
         * so the heap library can locate the 'hello there' string and
         * they are set with the location of the new string.
         */
        ret = diskheap_update(diskheap_p, "hello there again", 18, 0, block_n,
                              offset, 0, &block_n, &offset, 0L, 0L);
        /* ret should be checked against DISKHEAP_ERROR_NONE */

        /*
         * Lookup the block-number and offset in the heap.  This
         * returns an allocated buffer of memory containing the string
         * while the size variable is set with its length.
         */
        str_p = diskheap_retrieve(diskheap_p, block_n, offset, &size, 0L,
                &ret);
        /* str_p should be checked against NULL */

        printf("String '%s' (size %u) is at block #%u, offset %u\n",
```

```
                str_p, size, block_n, offset);
        free(str_p);
}
```

2.4 Some Ideas on How to Utilize the Library

This library was initially designed to provide the storage substrate for a high performance disk hash table library which I will be writing soon. It has been on my mind for some time however as I've pondered various projects which need underlying disk functionality. Flat files work efficiently for many applications however as soon as a program is adding, removing, resizing, updating, etc. transactions to any great degree, the Diskheap library should be considered.

Some ideas for using the library include tree structures, linked lists, skip lists, and virtual file systems. I encourage you to send me either projects where you have used the library or ideas for usage.

3 List of functions provided by the library.

The functions listed here are for learning purposes only and will *not* be as up to date as the '`diskheap.h`' header file. If you are writing your program, I'd encourage you to use it as a reference. All of the information in these function lists should be in the header file as well.

3.1 Standard Functions such as Open, Close, Store, and Retrieve.

3.1.1 diskheap_create – Create a new diskheap file

Function

diskheap_t *diskheap_create(const char **file*, const unsigned int *flags*, const unsigned int *block_size*, const unsigned int *heap_type*, const unsigned int *open_mode*, int **error_p*)

Usage: `diskheap_p = diskheap_create("stuff.dh", 0, 0 /* use default block-size */, 0 /* no heap type specified */, 0644, &ret /* error code */);`

This function creates a brand new diskheap file when the file has not existed before. It takes arguments similar to the `open` system call along with the heap-type which can be used by the caller to identify the contents of the heap.

3.1.2 diskheap_open – Open an existing diskheap file

Function

diskheap_t *diskheap_open(const char **file*, const unsigned int *flags*, unsigned int **heap_type_p*, int **error_p*)

Usage: `diskheap_p = diskheap_open("stuff.dh", 0 /* no flags */, 0L /* don't want the type */, &ret /* error code */);`

This function opens a Diskheap file that was created beforehand with `diskheap_create`. It takes a pointer to the heap-type variable which will be set to the number passed to `diskheap_create`.

3.1.3 diskheap_close – Close a diskheap structure

Function

int diskheap_close(diskheap_t **diskheap_p*)

Usage: `ret = diskheap_close(diskheap_p);`

This function closes a previously created or opened diskheap structure. It flushes any outstanding I/O, closes the file descriptor, and frees the memory in the structure. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

3.1.4 diskheap_store – Store a buffer of bytes in the heap

Function

int diskheap_store(diskheap_t *_diskheap_p_, const void *_buffer_, const unsigned int _user_size_, const unsigned int _user_type_, unsigned int *_block_num_p_, unsigned int *_offset_p_)

Usage: `ret = diskheap_store(diskheap_p, "hello there", 11 /* size of string */, 0 /* no type specified */, &block_num, &offset);`

This function stores a buffer of bytes into the diskheap returning the block-number and offset location where it was written. You will need to record the block-number and offset location information somewhere so you can retrieve or delete this space from the heap later. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

3.1.5 diskheap_retrieve – Retrieve a previously stored buffer

Function

void *diskheap_retrieve(diskheap_t *_diskheap_p_, const unsigned int _block_num_, const unsigned int _offset_, unsigned int *_size_p_, unsigned int *_type_p_, int *_error_p_);

Usage: `buf_p = diskheap_retrieve(diskheap_p, block_num, offset, &size, &type_code, &ret);`

This function looks up a block-number and offset location and allocates and returns a dynamic memory buffer with its contents. It passes back the size of the buffer in a size argument and the type that was passed to `diskheap_store` in a type argument. It will return 0L on an error and set the error code argument.

NOTE: you must deallocate the returned buffer with a call to `free()` at a later time. To use a static buffer instead, see `diskheap_retrieve_to_buf`.

3.1.6 diskheap_retrieve_to_buf – Retrieve into a fixed buffer.

Function

int diskheap_retrieve_to_buf(diskheap_t * _diskheap_p_, const unsigned int _block_num_, const unsigned int _offset_, void *_buffer_, const unsigned int _max_read_size_, unsigned int * _size_p_, unsigned int * _type_p_);

Usage: `ret = diskheap_retrieve_to_buf(diskheap_p, block_num, buffer, 1024 /* buffer size */, offset, &size, &type_code, &ret);`

This is the same as the `diskheap_retrieve` function but instead of allocating a buffer, it will use the buffer that it is passed. You can use fixed sized buffers that do not have to be allocated or freed with this function. Also, if you limit the size of the buffer, you can read in the first couple of bytes from it without reading in the entire stored entity. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

3.1.7 diskheap_update – Update a stored buffer with new data.

Function

int diskheap_update(diskheap_t * *diskheap_p*, const void **new_buffer*, const unsigned int *new_size*, const unsigned int *new_type*, const unsigned int *old_block_num*, const unsigned int *old_offset*, const int *safer_b*, unsigned int * *block_num_p*, unsigned int * *offset_p*, unsigned int * *old_size_p*, unsigned int * *old_type_p*);

Usage: `ret = diskheap_update(diskheap_p, "new string", 10 /* size of string */, 0 /* no type specified */, block_num, offset, 1 /* safer flag */, &new_block_num, &new_offset, &old_size, &old_type);`

This function replaces a stored item with a new item. You specify the new item's size and type and the block-number and offset location of the old item. It will return the new location of the new item and the size and type of the old item. This basically does a `diskheap_delete` and a `diskheap_store` is that order but if you specify 1 for the `safer` flag, then it will do the store first and then the delete. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

3.1.8 diskheap_delete – Remove a store buffer from the heap.

Function

int diskheap_delete(diskheap_t * *diskheap_p*, const unsigned int *block_num*, const unsigned int *offset*, unsigned int * *user_size_p*, unsigned int * *user_type_p*);

Usage: `ret = diskheap_update(diskheap_p, block_num, offset, &size, &type);`

This function deletes a previously stored item from the heap. You specify the old item's block-number and offset location and it passes back its size and type. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

3.2 Administrative Functions

The following administrative functions tune some of the internal settings and should not be necessary to call unless you are an experienced Diskheap programmer.

3.2.1 diskheap_set_free_space – Adjust Space for Free Information

Function

int diskheap_set_free_space(diskheap_t * *diskheap_p*, const unsigned int *free_space*);

Usage: `ret = diskheap_set_free_space(diskheap_p, 10240000);`

This function sets the amount of disk space to reserve for free-space information. The default is currently 1mb which should be able to store more than 150,000 free slots in the file. Each "free slot" represents a block-number and size (in blocks) of a free area in the diskheap. Contiguous free space is combined so each free area is bounded by allocated areas. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

Free slots which cannot be accounted for in this area will not be stored and may be lost however the default settings should make this a rare occurrence.

Note: this call has to be made *immediately* following the call to `diskheap_create` and no later. Once the file has been created, it cannot be adjusted.

3.2.2 diskheap_set_sync_often – When to Sync Administrative Information

Function

int diskheap_set_sync_often(diskheap_t * *diskheap_p*, const int *sync_header*, const int *sync_free*);

Usage: `ret = diskheap_set_sync_often(diskheap_p, 10000, 10000);`

This function sets how many Diskheap transactions must occur before the header and free-list administrative information should be written to disk. This administrative information needs to be up-to-date and should be updated every once in a while in case your program exits unexpectantly and does not close the Diskheap properly. You can set either argument to 0 to have the sync never happen unless you call the `diskheap_sync_header` and `diskheap_sync_free_list` functions. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

3.2.3 diskheap_sync_header – Sync Administrative Header Information

Function

int diskheap_sync_header(diskheap_t * *diskheap_p*);

Usage: `ret = diskheap_sync_header(diskheap_p);`

This function syncs the administrative header information from memory to disk. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

3.2.4 diskheap_sync_free_list – Sync Free-List Information.

Function

int diskheap_sync_free_list(diskheap_t * *diskheap_p*);

Usage: `ret = diskheap_sync_free_list(diskheap_p);`

This function syncs the administrative free-list information from memory to disk. The free-list records which space in the Diskheap is not currently in use and can be given out to future store operations. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

3.3 Functions to Associate String Labels with Locations

Since the diskheap allocations are stored at arbitrary locations in the heap, there is no way for a program to read from the front of a file to get to administrative information. To help a library know where to make it's "first read" from the diskheap, the library provides the ability to associate locations with a string label. Starting point for an on-disk data structure.

Examples of usage include associated the label '`hashstart`' with the location of the bucket information for a hash table. You could store the location of the first entry in a linked list with the label '`liststart`' and the last entry with '`listend`'. If you are implementing a mini-file system, you could associate the top directory location with the label '`root_directory`'.

3.3.1 diskheap_label_set – Associate a label with a diskheap location.

Function

int diskheap_label_set(diskheap_t * *diskheap_p*, const char **label*, const unsigned int *block_num*, const unsigned int *offset*, const int *overwrite_b*);

Usage: `ret = diskheap_label_set(diskheap_p, "start", block_num, offset, 1 /* overwrite */);`

This function associates a specific block-number and offset location with a string label. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

3.3.2 diskheap_label_get – Get the location associated with a label.

Function

int diskheap_label_get(diskheap_t * *diskheap_p*, const char **label*, unsigned int * *block_num_p*, unsigned int * *offset_p*);

Usage: `ret = diskheap_label_get(diskheap_p, "start", &block_num, &offset);`

This function gets the block-number and offset location that is associated with a specific string label. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

3.3.3 diskheap_label_get_entry – Get a specific entry from the label array.

Function

int diskheap_label_get_entry(diskheap_t * *diskheap_p*, const unsigned int *entry_n*, char ***label_p*, unsigned int * *block_num_p*, unsigned int * *offset_p*);

Usage: `ret = diskheap_label_get_entry(diskheap_p, 1 /* entry number */, &label_p, &block_num, &offset);`

There are a certain number (currently 10) of label and location associations stored in the diskheap header. This function gets a specific entry and returns the label string and the associated block-number and offset location. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

NOTE: The `label_p` string pointer argument must be passed to `free` to be deallocated.

3.4 Administrative Functions

3.4.1 diskheap_linear_first – Get the location of first entry in the heap.

Function

int diskheap_linear_first(diskheap_t * *diskheap_p*, diskheap_linear_t * *linear_p*);

Usage: `ret = diskheap_linear_first(diskheap_p, &linear);`

This function starts the linear access operation by setting the block-number and offset location to the first allocation found in the heap. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code including DISKHEAP_ERROR_NOT_FOUND if there are no entries in the heap.

3.4.2 diskheap_linear_next – Get the location of next entry in the heap.

Function

int diskheap_linear_next(diskheap_t * *diskheap_p*, diskheap_linear_t * *linear_p*);

Usage: `ret = diskheap_linear_next(diskheap_p, &linear);`

This function adjusts the linear structure to reference the next allocation location in the heap. You can start at any valid location in the heap. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code including DISKHEAP_ERROR_NOT_FOUND if there are no more entries in the heap.

3.5 Miscellaneous Functions

3.5.1 diskheap_fsync – Sync the diskheap with disk using fsync.

Function

int diskheap_fsync(diskheap_t * *diskheap_p*);

Usage: `ret = diskheap_fsync(diskheap_p);`

This function calls fsync on the file descriptor associated with the Diskheap. It is designed to make sure that all buffered data gets moved to the disk. See the manual entry for `fsync` to determine what this does in reality. This It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

Warning: even if the fsync succeeds, there is no guarantee that if the system would immediately crash that the diskheap file would not be corrupted.

Warning: on some operating systems, fsync may not be available in which case the DISKHEAP_ERROR_OS_CAPABLE error code will be returned.

3.5.2 diskheap_seed_random – Seed the random number generator.

Function

void diskheap_seed_random(const int *seed*);

Usage: `diskheap_seed_random(time(0L) ^ getpid());`

This function seeds the random number generator inside of the diskheap library and sets a static flag which will not cause any more calls to the random seed routine to be made. This should be called before any other diskheap calls are made to be effective. It will return DISKHEAP_ERROR_NONE if it succeeds otherwise an error code.

Note: this is usually only needed in testing since the library auto seeds the random number generator the first time a diskheap is opened or created.

3.5.3 diskheap_strerror – Return string equivalent to diskheap error.

Function

const char *diskheap_strerror(const int *error*);

Usage: `printf("diskheap_open failed and returned: %s\n", diskheap_strerror(ret));`

This function returns the string version of a Diskheap error codes. It is useful if you want to log a Diskheap error code.

4 Index of Concepts

Table of Contents