mtree parsing and manipulation library

Michal Ratajsky
Palacky University, Czech Republic
michal@FreeBSD.org

***Abstract*—FreeBSD includes several programs which create or consume file system hierarchy descriptions in the *mtree(5)* format. These descriptions, also called specifications, have a broad range of uses, from automatically creating directory structures to security auditing.**

Each of the programs, namely *mtree(8)*, *bsdtar(1)*, *install(1)* and *makefs(8)*, has its own implementation of the mtree format. This not only adds maintenance overhead, but also makes interoperability difficult, as each of the implementations only supports a limited subset of the format.

In 2015, the *libmtree* library was created as part of the Google Summer of Code program to replace and unify the existing implementations.

I. MTREE SPECIFICATION

An *mtree specification* is a textual description of file system objects. A single line in a specification describes a single file along with its properties and is also referred to as *mtree entry*. Properties are represented by *keywords*, indicating property names, and values as demonstrated in Listings 1 and 2. These also demonstrate the differences between the two most common mtree specification formats.

```
.               type=dir
    adjkerntz   type=file size=9280
    atmconfig   type=file size=60392
    badsect     type=file size=8528
    bsdlabel    type=file size=36928
    camcontrol  type=file size=138128
    ccdconfig   type=file size=12648
    clri        type=file size=7240
    comcontrol  type=file size=7208
    conscontrol type=file size=8472
    ddb         type=file size=12120
    devd        type=file size=1107624
    devfs       type=file size=15656
    dhclient    type=file size=95888
```

Listing 1. An mtree specification in the classic (version 1) format

```
. type=dir
./adjkerntz type=file size=9280
./atmconfig type=file size=60392
./badsect type=file size=8528
./bsdlabel type=file size=36928
./camcontrol type=file size=138128
./ccdconfig type=file size=12648
./clri type=file size=7240
./comcontrol type=file size=7208
./conscontrol type=file size=8472
./ddb type=file size=12120
./devd type=file size=1107624
./devfs type=file size=15656
./dhclient type=file size=95888
```

Listing 2. An mtree specification in the long (version 2) format

Specifications are created and processed by various tools to store records of changes done to a file system, create directory structures or verify file system integrity.

II. IMPLEMENTATIONS IN FREEBSD

The following programs are included in FreeBSD and contain mtree implementations:

- the *mtree(8)* program serves as a comprehensive tool for creating specifications for file hierarchies, manipulating file systems according to specifications as well as performing auditing actions;
- the *install(1)* program is capable of producing mtree specifications containing lists of installed items;
- the *makefs(8)* program is able to create file system images according to mtree manifests;
- the *libarchive(3)* library and *bsdtar(1)* program include support for reading and writing mtree specifications.

All of these programs contain unique mtree implementations, written by different authors and with varying degree of completeness and different handling of non-standard cases. The differences between implementations create interoperability problems, where a specification created by one implementation might not be parsed by another implementation or might be interpreted erroneously. Also, differences in handling of non-standard and error conditions may lead to unexpected behavior, e.g. an unknown keyword may cause an implementation to either signal an error condition or be silently ignored.

III. THE LIBMTREE PROJECT

To address the aforementioned issues, a FreeBSD Google Summer of Code project idea was proposed by Brooks Davis [1]. The project idea suggested creating an independent library, capable of replacing the existing mtree implementations and also flexible enough to support future development and extensions.

The project proposal, selected for the Google Summer of Code 2015 program, was further expanded to include porting of all programs and libraries present in FreeBSD to use the newly created *libmtree* library, replacing their original mtree implementations.

Additionally, it was decided that *libmtree* would become an independent project and be written with portability in mind. The *mtree(8)* program would be updated to improve its portability and included in the *libmtree* source tree.

No external dependencies were supposed to be required to build and use the library for portability reasons and to allow inclusion in the FreeBSD source tree. In a later stage

of development, *libmtree* gained the ability to use the *libnettle* library to support hash functions that are not provided by the operating system.

IV. API DESIGN

The libmtree API was designed to accommodate the needs of existing applications, as well as provide additional functions to allow greater application development flexibility and future extensions.

The provided interfaces can be divided into three major components:

- the higher-level *mtree_spec* interface, provided for manipulating complete specifications;
- the lower-level *mtree_entry* interface, allowing access to individual entries contained in specifications;
- minor interfaces, provided for convenience or manipulating more complex data types contained in entries, see section IV-C below.

The *mtree_spec* interface is capable of parsing and creating mtree specifications in all the common formats:

- the classic (version 1) format as shown in Listing 1;
- the long (version 2) format as shown in Listing 2;
- the long (version 2) format with file path after keywords, as produced by the *"mtree -D"* command.

The existing programs also require the ability to traverse entries inside specifications and access/manipulate their keyword values. Some of the original implementations arrange mtree entries into a tree structure that resembles their path hierarchy. *libmtree* uses linked lists of entries instead as it attempts to accurately capture the actual content of specifications and preserve original ordering. Radix trees are used internally to improve look up times in certain operations, but these are currently not exposed in the API.

Several additional features have been implemented, such as comparison, merging, read/write filtering and arbitrary sorting of entries. Some of these have already been used in application code, while others have been provided for completeness and greater flexibility.

A. Introduction to the mtree_spec interface

The *mtree_spec* interface mainly provides support for reading and writing complete specifications and also serves as a lightweight container of mtree entries.

```
FILE *fp;
struct mtree_spec *spec;

fp = fopen("/etc/mtree/BSD.var.dist", "r");

spec = mtree_spec_create();
mtree_spec_read_spec_file(spec, fp);
```

Listing 3. Reading a specification from a file.

It is also possible to supply path to a directory and have *libmtree* traverse the file hierarchy and add some or all items to a specification.

Any of the reading functions can be called any number of times to accumulate entries into a single specification. Reading can be further configured by selecting which keywords to read and by providing reading options.

```
mtree_spec_set_read_path_keywords(spec,
        MTREE_KEYWORD_TYPE |
        MTREE_KEYWORD_UID |
        MTREE_KEYWORD_SIZE);

/*
 * Sort the entries after each call of a
 * reading function and merge duplicates.
 */
mtree_spec_set_read_options(spec,
        MTREE_READ_SORT |
        MTREE_READ_MERGE);

/*
 * Traverse the directories and add
 * everything to the specification.
 */
mtree_spec_read_path(spec, "/bin");
mtree_spec_read_path(spec, "/sbin");
```

Listing 4. Appending directories to a specificiation.

Specification can be written in any format supported by the original *mtree(8)* tool. It is possible to specify write options to further customize the output. Using various combinations of these options allows creating output compatible with the existing programs.

```
FILE *fw = fopen("output.mtree", "w");

/*
 * Write spec in the long (version 2) format.
 */
mtree_spec_set_write_format(spec,
        MTREE_FORMAT_2_0);
mtree_spec_write_file(spec, fw);
```

Listing 5. Writing a specification.

B. Introduction to the mtree_entry interface

The *mtree_entry* interface is a lower-level interface intended for accessing individual entries and traversing and manipulating lists of entries, which are or may eventually be included in a specification.

An *mtree_entry* represents a single item, but also contains pointers to the previous and next items in sequence; it is a doubly-linked list. The library does not use any existing list implementation (such as *queue(3)*), but employs its own set of functions to access, manipulate, copy, search, merge and delete lists of mtree entries, as well as individual items.

```
struct mtree_entry *ent;

/*
 * Retrieve list of entries from a spec and
 * print out their file paths.
 */
for (ent = mtree_spec_get_entries(spec);
     ent != NULL;
     ent = mtree_entry_get_next(ent)) {
        printf("file %s\n",
                mtree_entry_get_path(ent));
}
```

Listing 6. Traversing a list of entries contained in an *mtree_spec*.

```
ent = mtree_entry_create("/bin/sh");

/*
 * libmtree can stat(2) the file and read
 * the properties by itself.
 */
mtree_entry_set_keywords(ent,
        MTREE_KEYWORD_TYPE |
        MTREE_KEYWORD_SIZE, 0);

/*
 * Or they can be set manually.
 */
mtree_entry_set_type(ent, MTREE_ENTRY_FILE);
mtree_entry_set_size(ent, 123);
```

Listing 7. Creating an entry and including some keywords.

Functions are provided to create linked lists of entries by appending, prepending, or inserting entries at arbitrary positions.

```
struct mtree_entry *list = NULL;

/*
 * The common approach to create a list
 * of entries in a particular order.
 */
list = mtree_entry_prepend(list, entry1);
list = mtree_entry_prepend(list, entry2);
list = mtree_entry_prepend(list, entry3);

list = mtree_entry_reverse(list);
```

Listing 8. Creating a list of entries.

C. Other interfaces

The *mtree_spec_diff* interface compares two existing specifications and produces lists of entries which are present in only one of the specifications, or where keyword values do not match.

These lists can be accessed directly or written out in the *"mtree -f -f"* format.

```
mtree_spec_read_spec_file(spec1, fp1);
mtree_spec_read_spec_file(spec2, fp2);

struct mtree_spec_diff *diff;

/*
 * Compare all keywords.
 */
diff = mtree_spec_diff_create(spec1,
        spec2,
        MTREE_KEYWORD_MASK_ALL, 0);
/*
 * Write differences to a file stream.
 */
mtree_spec_diff_write_file(diff, fw);
```

Listing 9. Comparing specifications.

Other interfaces include:

- *mtree_cksum* for calculating POSIX 1003.2 checksum for use with the *cksum* keyword,
- *mtree_digest* for calculating digests of various types,
- *mtree_device* for accessing and manipulating values of the *device* and *resdevice* keywords.

V. Final Results

The project was mentored by Brooks Davis and successfully implemented as part of the Google Summer of Code 2015 program.

As of February 2016, the project has not yet been integrated into FreeBSD and an integration plan has yet to be developed. However there are ongoing efforts to provide the project with testing, integration work and solve the known issues.

Source code is available in public GitHub repositories [2] [3] [4] and further information, along with a summary of known issues, is available on the project wiki page [5].

References

[1] FreeBSD Summer of Code Ideas: mtree parsing and manipulation library; https://wiki.freebsd.org/SummerOfCodeIdeas#mtree_parsing_and_manipulation_library.
[2] GitHub, libmtree; https://github.com/mratajsky/libmtree.
[3] GitHub, libarchive; https://github.com/mratajsky/libarchive.
[4] GitHub, freebsd; https://github.com/mratajsky/freebsd.
[5] mtree parsing and manipulation library, FreeBSD Wiki; https://wiki.freebsd.org/SummerOfCode2015/mtreeParsingLibrary.