# Container Library and FUSE Container File System

## Softwarepraktikum für Fortgeschrittene

Michael Kuhn

Parallele und Verteilte Systeme
Institut für Informatik
Ruprecht-Karls-Universität Heidelberg

2008-02-29

- An attempt to decrease the metadata overhead is to maintain a reduced set of metadata files
- This usually has to be done manually
  - Available file systems do not permit the user to change which metadata is stored
- One approach is to pack them together in one file – a container
  - The file system only manages metadata for one file
- Within this container, the files and their corresponding metadata can be managed arbitrarily

- The container format should enable random access to provide access times independent of the position of a file within the container
- Existing formats are insufficient
    - The `tar` format does not provide random access
    - The `iso` format stores too much metadata
- Therefore a new container format was designed and implemented by Hendrik Heinrich
- This existing implementation was used as a basis for all work presented here

- The container library was not licensed in any way
    - This made future development and usage difficult
- It has been licensed under a 2-clause BSD license in agreement with the original author

- The library was completely overhauled
  - It now provides consistently named functions and data types
- The comments within the code were modified to allow the automatic generation of an API documentation with Doxygen

## Architecture Independence

- The original implementation did not honor the different sizes of data types on 32 and 64 bit architectures
  - This made it impossible to use a container created with a 32 bit version of the library with a 64 bit version and vice versa
- By using datatypes of a fixed size the containers can now be used on both 32 and 64 bit architectures without problems
- The size of the metadata structures used in the library is now independent of the architecture
  - C pads structures for better memory alignment

## Thread-Safety

- The library was not thread-safe in its original form
  - It uses a shared file pointer for all files in a container
  - The functions `read` and `write` were used in combination with `lseek`
- The library was modified to use the `pread` and `pwrite` functions that do not modify the shared file pointer
- It can now be used safely in multi-threaded applications

## Write Support

- The original version of the container library lacked an easy-to-use method to create containers
- A convenient interface to add new files was added to the library
  - Either from memory with `ct_file_create_buffer`
  - Or from an existing file with `ct_file_create_path`
- There are currently also "fast" versions of these functions

## File Hashing

- File data stored in the container is protected from silent corruption
- Currently a SHA-1 hash of it is stored along with its metadata
- This also needs to be done for the metadata

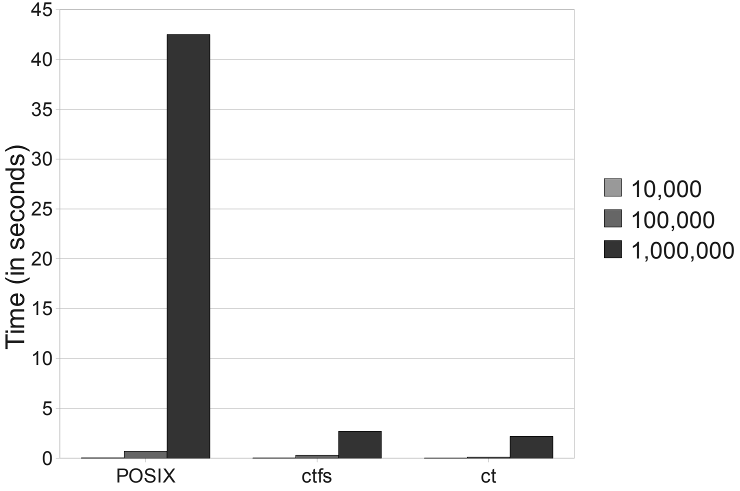
- Five tools to work with containers on the command line
  - `ctcat`
  - `ctcp`
  - `ctls`
  - `ctmk`
  - `ctpc`

- `ctcat`
  - Print the contents of a file in a container to `stdout`
- `ctcp`
  - Copy a file from a container to a local file system
- `ctls`
  - Print the names of all files in a container
  - Optionally also print their hashes or sizes
- `ctmk`
  - Create a new container with all files in a given directory
- `ctpc`
  - Copy a file from a local file system to a container

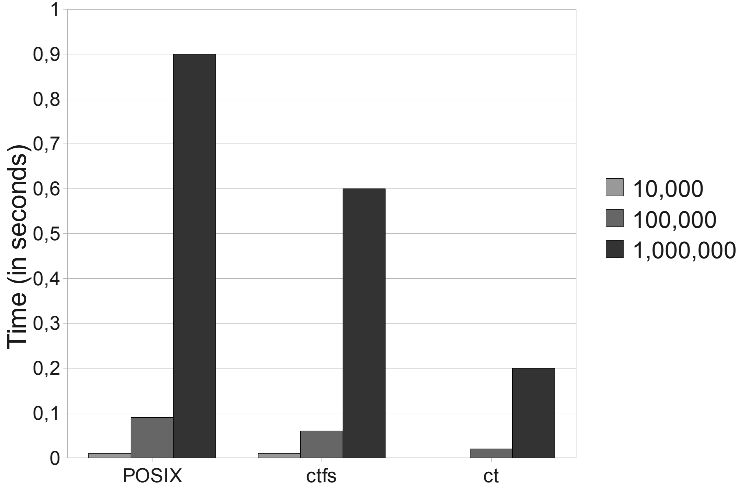
- There are legacy applications that use the POSIX API and can not be ported
  - It may be too much work to port them
  - Or the source code is not available at all
- Provide POSIX access to containers
- The easiest way to do this is to write a FUSE file system

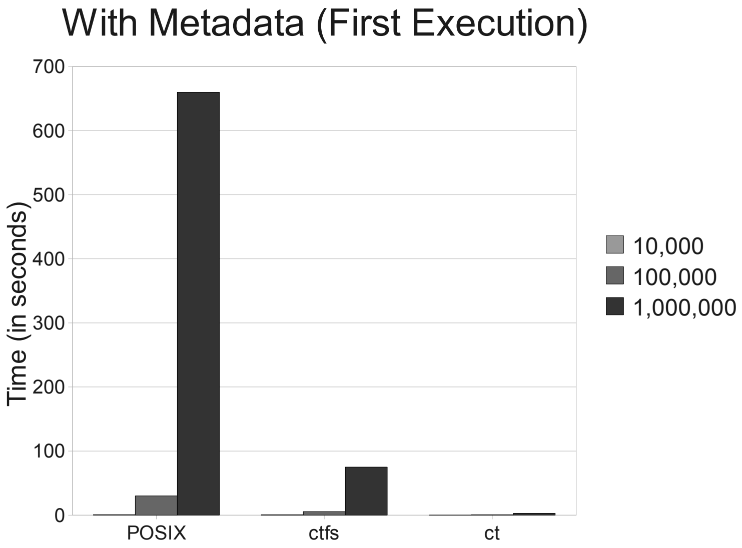
- Provide an overlay file system
  - The whole underlying file system is accessible
  - Directories are handled normally – as directories
  - Files are handled as containers – that is, also as directories
- For example
  - `ctfs` is mounted at `/ctfs`
  - There is a container available as `/storage/stuff.ct`
  - All files within this container would be available in the directory `/ctfs/storage/stuff`
- There is more room for optimization
  - Does two `stat()` calls for each container file
  - One can be eliminated easily

## Without Metadata (First Execution)

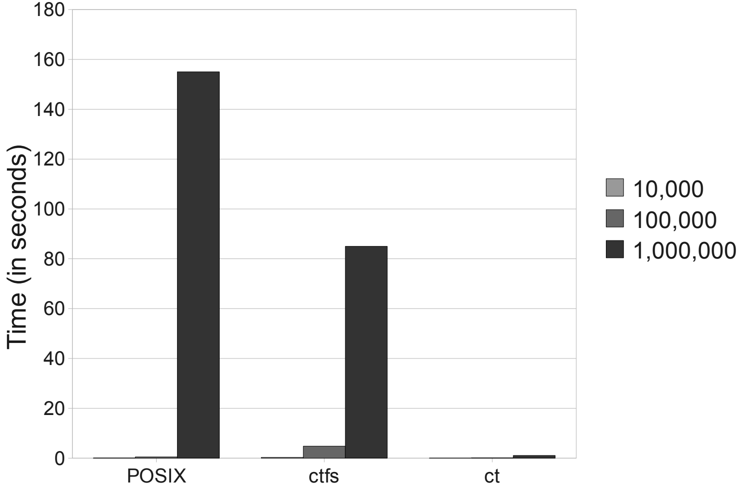


## Without Metadata




With Metadata (First Execution)
Time (in seconds)
0
100
200
300
400
500
600
700
POSIX
ctfs
ct
10,000
100,000
1,000,000

## With Metadata


Time (in seconds)
POSIX
ctfs
ct