RADB

Raja Mukherji

May 12, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

There are already numerous embedded key-value stores available, where the keys are null-terminated strings or arbitrary byte sequences. *RADB* (*Raja's Attempt at a Database*) is a small *C* library that for using integer-indexed value stores with either fixed or variable length data.

In addition *RADB* provides lookup tables for keys of either fixed or variable length, returning integer indices which can be used with the integer-indexed value stores (if desired).

CHAPTER

TWO

BUILDING

Note: Currently *RADB* has been tested on *Linux*, *FreeBSD* and *MacOS*. It has no dependencies other than `mmap()`. Being developed primarily on *Linux*, there may be some unintentional dependencies on *Glibc*, patches are welcome.

```
$ git clone https://github.com/rajamukherji/radb
$ cd radb
$ make [RADB_MEM=<MALLOC | GC>]
$ make install [PREFIX=<install path>]
```

CHAPTER

THREE

API

In order to be as versitile as possible, *RADB* support different memory management configurations when built. Building with `make RADB_MEM=MALLOC` or `make RADB_MEM=GC` will use the system `malloc()` or `GC_malloc()` from the Hans-Boehm garbage collector respectively. Omitting the `RADB_MEM` flag when building defaults to the user specified memory allocator per store. The *RADB_MEM_PARAMS* is defined accordingly:

RADB_MEM_PARAMS
: This macro is empty if `RADB_MEM` is defined as `MALLOC` or `GC` when *RADB* is built. Otherwise it defines the following additional parameters in functions for creating or opening any store or index.

  void *Allocator User-data for the allocator.

  void *(*alloc)(void *Allocator, size_t Size) Allocate `Size` bytes of memory with the allocator.

  void *(*alloc_atomic)(void *Allocator, size_t Size) Allocate `Size` bytes of pointer-free memory with the allocator.

  void (*free)(void *Allocator, void *Ptr) Free memory at `Ptr` from the allocator.

3.1 Stores

RADB stores allow values to be stored and retrieved using integer indices. Every index (currently limited to 2^{32} - 2) is considered valid, stores will grow automatically when writing to an index beyond the current store size and return an empty value (for string stores) when reading beyond the current store size. This means users are free to decide how to allocate indices in a store. For convenience, stores also provide a chain based index allocator to maintain a free chain of indices.

> **Warning:** The free index chain must not be used with user allocated indices in the same store as the chain is maintained in the value storage and will be corrupted if a value is set without being allocated from the chain.

3.1.1 String Store

String stores can contain arbitrary byte sequences (including null bytes). The values are split into blocks, the size of each block is set when the store is created. Each string store consists of two files: an *entries* file containing the length and initial block of each value in the store and a *data* file containing the blocks. Both files are created relative to the prefix path provided when a string store is created.

struct **string_store_t**
: A string store.

string_store_t *string_store_create(const char *Prefix, size_t RequestedSize, size_t ChunkSize RADB_MEM_PARAMS)

Creates a new string store.

Returns An open string store.

Parameters

- **Prefix** – The prefix path for the store.
- **RequestedSize** – The requested size for each block (rounded up to the nearest power of 2).
- **ChunkSize** – The additional number of bytes to allocate each time the entries or data file needs to grow (rounded up to the nearest multiple of 512).
- **RADB_MEM_PARAMS** – Additional parameters if per-store memory allocation is enabled, see *RADB_MEM_PARAMS*.

string_store_t *string_store_open(const char *Prefix RADB_MEM_PARAMS)

Opens an existing string store. Returns NULL if an error occurs.

Returns An open string store.

Parameters

- **Prefix** – The prefix path for the store.

void **string_store_close**(*string_store_t* *Store)

Closes an open string store.

Parameters

- **Store** – An open string store.

size_t **string_store_num_entries**(*string_store_t* *Store)

Returns The number of entries allocated in the store. Note that this is an upper bound and is usually larger than the actual number of entries set in the store.

Parameters

- **Store** – An open string store.

size_t **string_store_size**(*string_store_t* *Store, size_t Index)

Returns

The size of the Index-th value. If no value has been set at Index then 0 is returned.

param Store An open string store.

param Index A valid index.

size_t **string_store_get**(*string_store_t* *Store, size_t Index, void *Buffer, size_t Space)

Gets at most Space bytes from the Index-th value into Buffer.

Returns The number of bytes copied.

Parameters

- **Store** – An open string store.
- **Index** – A valid index.
- **Buffer** – A buffer of at least Space bytes.

- **Space** – The number of bytes to get.

void **string_store_set**(*string_store_t* *Store, size_t Index, const void *Buffer, size_t Length)
Sets the `Index`-th value to contents of `Buffer`.

Parameters

- **Store** – An open string store.
- **Index** – A valid index.
- **Buffer** – A buffer of at least `Length` bytes.
- **Length** – The number of bytes to set.

int **string_store_compare**(*string_store_t* *Store, const void *Other, size_t Length, size_t Index)
Convenience function to compare (lexicographically) the `Index`-th value to another value without copying.

Returns

- -1 if `Other` occurs before the `Index`-th value.
- 1 if `Other` occurs after the `Index`-th value.
- 0 if `Other` is the same as the `Index`-th value.

Parameters

- **Store** – An open string store.
- **Other** – A buffer of at least `Length` bytes.
- **Length** – The number of bytes to compare (at most).
- **Index** – A valid index.

int **string_store_compare2**(*string_store_t* *Store, size_t Index1, size_t Index2)
Convenience function to compare (lexicographically) the `Index1`-th value to the `Index2`-th value without copying.

Returns

- -1 if the `Index1`-th value occurs before the :c:`Index`2-th value.
- 1 if the `Index1`-th value occurs before the :c:`Index`2-th value.
- 0 if the `Index1`-th value is the same as the :c:`Index`2-th value.

Parameters

- **Store** – An open string store.
- **Index1** – A valid index.
- **Index2** – A valid index.

size_t **string_store_alloc**(*string_store_t* *Store)
Allocates an index from the free index chain.

Returns A valid index

Parameters

- **Store** – An open string store.

void **string_store_free**(*string_store_t* *Store, size_t Index)
Returns an index to the free index chain.

Parameters

- **Store** – An open string store.
- **Index** – A valid index (from a previous call to *string_store_alloc()*).

struct **string_store_writer_t**
For writing to a value in a string store in a stream.

void **string_store_writer_open**(*string_store_writer_t* *Writer, *string_store_t* *Store, size_t Index)
Prepares a *string_store_writer_t* for writing to the `Index`-th value. There is no need to close a writer. Writers may be reused by calling this function again.

Parameters

- **Writer** – A writer.
- **Store** – An open string store.
- **Index** – A valid index.

size_t **string_store_writer_write**(*string_store_writer_t* *Writer, const void *Buffer, size_t Length)
Writes bytes from `Buffer` to the selected value in the store.

Returns The number of bytes written, which will always be `Length` (excluding errors).

Parameters

- **Writer** – An open writer.
- **Buffer** – A buffer of at least `Length` bytes.
- **Length** – The number of bytes to write.

struct **string_store_reader_t**
For reading the bytes of a value in a string store in a stream.

void **string_store_reader_open**(*string_store_reader_t* *Reader, *string_store_t* *Store, size_t Index)
Prepares a *string_store_reader_t* for reading from the `Index`-th value. There is no need to close a reader. Readers may be reused by calling this function again.

size_t **string_store_reader_read**(*string_store_reader_t* *Reader, void *Buffer, size_t Length)
Reads bytes from the selected valuei in the store into `Buffer`.

Returns The number of bytes read. May be 0 if the end of the value has been reached.

Parameters

- **Reader** – An open reader.
- **Buffer** – A buffer of at least `Length` bytes.
- **Length** – The number of bytes to read (at most).

3.1.2 Fixed Store

struct **fixed_store_t**

fixed_store_t *fixed_store_create(const char *Prefix, size_t RequestedSize, size_t ChunkSize RADB_MEM_PARAMS)

fixed_store_t *fixed_store_open(const char *Prefix RADB_MEM_PARAMS)

void **fixed_store_close**(*fixed_store_t* *Store)

size_t **fixed_store_num_entries**(*fixed_store_t* *Store)

void ***fixed_store_get**(*fixed_store_t* *Store, size_t Index)

size_t **fixed_store_alloc**(*fixed_store_t* *Store)

void **fixed_store_free**(*fixed_store_t* *Store, size_t Index)

3.2 Indices

3.2.1 String Index

struct **string_index_t**

string_index_t *string_index_create(const char *Prefix, size_t KeySize, size_t ChunkSize RADB_MEM_PARAMS)

string_index_t *string_index_open(const char *Prefix RADB_MEM_PARAMS)

size_t **string_index_count**(*string_index_t* *Store)

void **string_index_close**(*string_index_t* *Store)

size_t **string_index_insert**(*string_index_t* *Store, const char *Key, size_t Length)

size_t **string_index_search**(*string_index_t* *Store, const char *Key, size_t Length)

struct **string_index_result_t**

size_t **Index**

int **Created**

string_index_result_t **string_index_insert2**(*string_index_t* *Store, const char *Key, size_t Length)

size_t **string_index_size**(*string_index_t* *Store, size_t Index)

size_t **string_index_get**(*string_index_t* *Store, size_t Index, void *Buffer, size_t Space)

size_t **string_index_delete**(*string_index_t* *Store, const char *Key, size_t Length)

3.2.2 Fixed Index

struct **fixed_index_t**

fixed_index_t *fixed_index_create(const char *Prefix, size_t KeySize, size_t ChunkSize RADB_MEM_PARAMS)

fixed_index_t *fixed_index_open(const char *Prefix RADB_MEM_PARAMS)

size_t **fixed_index_count**(*fixed_index_t* *Store)

void **fixed_index_close**(*fixed_index_t* *Store)

size_t **fixed_index_insert**(*fixed_index_t* *Store, const char *Key)

size_t **fixed_index_search**(*fixed_index_t* *Store, const char *Key)

struct **fixed_index_result_t**

size_t **Index**

int **Created**

fixed_index_result_t **fixed_index_insert2**(*fixed_index_t* *Store, const char *Key)

const void ***fixed_index_get**(*fixed_index_t* *Store, size_t Index)

size_t **fixed_index_delete**(*fixed_index_t* *Store, const char *Key)

CHAPTER

FOUR

INDEX

INDEX