The cdb-reading library interface

Overview

You can read records in a constant database from file descriptor *fd* as follows:

1. Use `cdb_init` to place information about *fd* into a `struct cdb` variable *c*.
2. Carry out any number of searches, as described below.
3. Use `cdb_free` to remove any memory map that might have been reserved by `cdb_init`.

Each search works as follows:

1. Use `cdb_find` to search for a record under key *k*. If `cdb_find` returns 0, the database does not contain that key; stop. If `cdb_find` returns -1, there was a read error; abort.
2. Use `cdb_datalen` to find the number of bytes of data in this record. Allocate a pointer *d* to a region of memory large enough to hold the data. If not enough memory is available, abort.
3. Use `cdb_read` with `cdb_datapos` to read the data. If `cdb_read` returns -1, there was a read error; abort.
4. Do something with the data, and then free the allocated region of memory.

There may be several records under a single key. You can use `cdb_findnext` to find the next record under this key.

Details

```
#include <cdb.h>

cdb_init(&c,fd);
cdb_free(&c);
result = cdb_read(&c,d,dlen,dpos);

cdb_findstart(&c);
result = cdb_findnext(&c,k,klen);
result = cdb_find(&c,k,klen);

dpos = cdb_datapos(&c);
dlen = cdb_datalen(&c);

static struct cdb c;
int fd;

char *d;
unsigned int dlen;
uint32 dpos;

char *k;
unsigned int klen;
int result;
```

A `struct cdb` variable such as *c* is either unallocated or allocated. If it is allocated, it holds information about a constant database:

- a file descriptor *fd* reading from the database;

- if convenient, a shared memory map reading from the database; and
- information about a search in progress.

c must be initialized to zero, meaning unallocated.

`cdb_free` unallocates *c* if *c* is allocated. Otherwise it leaves *c* alone. `cdb_free` does not close *fd*.

`cdb_init` allocates *c* to hold information about a constant database read by descriptor *fd*. You may call `cdb_init` repeatedly; if *c* is already allocated, `cdb_init` unallocates it first.

`cdb_read` reads *dlen* bytes into *d* from byte position *dpos* in the database. You must allocate *c* before calling `cdb_read`. Normally `cdb_read` returns 0. If the database file is shorter than *dpos*+*dlen* bytes, or if there is a disk read error, `cdb_read` returns -1, setting `errno` appropriately.

`cdb_findstart` prepares *c* to search for the first record under a new key. You must allocate *c* before calling `cdb_findstart`, and you must call `cdb_findstart` before calling `cdb_findnext`.

`cdb_findnext` looks for the *n*th record under key *k* in the database, where *n* is the number of calls to `cdb_findnext` after the most recent call to `cdb_findstart`. If it finds the record, `cdb_findnext` returns 1; if there are exactly *n*`-1` such records, `cdb_findnext` returns 0; if there are fewer than *n*`-1` such records, the behavior of `cdb_findnext` is undefined; if there is a database format error or disk error, `cdb_findnext` returns -1, setting `errno` appropriately. Each call to `cdb_findnext` (before another call to `cdb_findstart`) must use the same *k* and *klen*.

If `cdb_findnext` returns 1, it arranges for `cdb_datapos` to return the starting byte position of the data in the record, and for `cdb_datalen` to return the number of bytes of data in the record. Otherwise the results of `cdb_datapos` and `cdb_datalen` are undefined.

`cdb_find` is the same as `cdb_findstart` followed by `cdb_findnext`: it finds the first record under key *k*.

Beware that these functions may rely on non-atomic operations on the *fd* ofile, such as seeking to a particular position and then reading. Do not attempt two simultaneous database reads using a single ofile.