TinyCDB - a Constant DataBase

by Michael Tokarev, mjt+cdb {at} tls {dot} msk {dot} ru.

Quick links

Introduction

`TinyCDB` is a very fast and simple package for creating and reading constant data bases, a data structure introduced by Dan J. Bernstein in his cdb package. It may be used to speed up searches in a sequence of (key,value) pairs with very big number of records. Example usage is indexing a big list of users - where a search will require linear reading of a large /etc/passwd file, and for many other tasks. It's usage/API is similar to ones found in BerkeleyDB, gdbm and traditional *nix dbm/ndbm libraries, and is compatible in great extent to `cdb-0.75` package by Dan Bernstein.

CDB is a *constant* database, that is, it cannot be updated at a runtime, only rebuilt. Rebuilding is atomic operation and is very fast - much faster than of many other similar packages. Once created, CDB may be queried, and a query takes very little time to complete.

Programming interface

There are two interfaces provided by a library, -lcdb, -- create interface which is used to create CDB file, and two variants of query interface. A program using any routines should #include <cdb.h> header file which holds all required definitions of a library. More information together with detailed description of every routine is available in manual page inside TinyCDB package.

TinyCDB is different from Dan's cdb-0.75 in the following ways:

- Only pure object-oriented approach is implemented, i.e. there is no documented way to directly write to or read from a CDB file. I found this approach more clean. With this, a record should fit in memory, but I don't know an application which may have a problem with that. All other hashing packages works the same way.
- Two query interfaces are provided - one compatible with Dan's cdb-0.75, and another which is compatible with older releases of his cdb library (which in fact allows direct reading of CDB file).
- At a time of CDB creation (cdb_make), it is possible to determine whenever a given key already exists (but this slows operation down, sometimes significantly), and to replace a key with a new one.
- TinyCDB usually requires much less memory to create CDB file, and never more than cdb-0.75.
- There is only one library, -lcdb, which required for linking and incorporates all the functionality (no additional libraries are required), and only one header file, <cdb.h>, which includes definitions for both cdb and cdb_make (it may be linked with cdb_make.h for programs that expects this header file).
- There is only one utility, `cdb`, which may be used for all the tasks including create, query, dump and so on. It understands Dan `cdbmake`'s input format, and traditional format understood by e.g. `makedb` utility found in BerkeleyDB package.

Create interface

Create interface is built around `struct cdb_make` structure which is opaque type. The following is a sequence of action which should be performed in order to create CDB file (error handling is omitted):

```
struct cdb_make cdbm;
int fd;
char *key, *val;
unsigned klen, vlen;

fd = open(tmpfile, O_RDWR|O_CREAT); /* open temporary file */
cdb_make_start(&cdbm, fd); /* initialize structure */

cdb_make_add(&cdbm, key, klen, val, vlen)
/* add as many records as needed */

cdb_make_put(&cdbm, key, klen, val, vlen, flag);
/* alternative interface.  flags is one of:
  CDB_PUT_ADD adds new record unconditionally like cdb_make_add()
  CDB_PUT_REPLACE if a key is already exists, replace the record
  CDB_PUT_INSERT  add a record only if the key isn't already exists
 */

cdb_make_exists(&cdbm, key, klen);
/* a routine to test whenever a given key is already exists */

cdb_make_finish(&cdbm);
/* final stage - write indexes to CDB file */

rename(tmpfile, cdbfile);
/* atomically replace CDB file with newly built one */
```

Query interface

There are two variants of query interface, one as found in cdb-0.75, and another as found in earlier versions of cdb (cdb-0.6x).

Query interface 1

This interface is built around `struct cdb` structure which is opaque to the application. This interface designed to be efficient for many queries, for a single query second variant may be more efficient. The following is a sequence of calls needed to perform a query of a value in a CDB file:

```
int fd;
struct cdb cdb;
char *key, *val;
unsigned klen, vlen, vpos;

fd = open(cdbfile, O_RDONLY);
cdb_init(&cdb, fd); /* initialize internal structure */
if (cdb_find(&cdb, key, klen) > 0) { /* if search successeful */
  vpos = cdb_datapos(&cdb); /* position of data in a file */
  vlen = cdb_datalen(&cdb); /* length of data */
  val = malloc(vlen); /* allocate memory */
  cdb_read(&cdb, val, vlen, vpos); /* read the value into buffer */
  ... /* handle the value */
}
```

and here is what is needed to enumerate all values assotiated with a given key:

```
struct cdb_find cdbf; /* structure to hold current find position */
cdb_findinit(&cdbf, &cdb, key, klen); /* initialize search of key */
while(cdb_findnext(&cdbf) > 0) {
```

```
  vpos = cdb_datapos(&cdb);
  vlen = cdb_datalen(&cdb);
  val = malloc(vlen);
  cdb_read(&cdb, val, vlen, vpos);
  /* handle the value */
  free(val);
}
```

Query interface 2

Another, simpler query interface exists which is sutable for a single query. Two routines provided works with a single filedescriptor opened for reading:

```
int fd;
char *key, *val;
unsigned klen
cdbi_t vlen;

fd = open(cdbfile, O_RDONLY); /* open a CDB file */
if (cdb_seek(fd, key, klen, &vlen) > 0) {
  /* if key was found, file will be positioned to the
   * start of data value and it's length will be placed to vlen */
  val = malloc(vlen);
  cdb_bread(fd, val, len); /* read the value;
                            * plain read() will do as well. */
  /* handle the value */
}
```

Format of CDB file

To be written. Meanwhile, consult Dan Bernstain's cdb manual.

Terms of usage

The code is in public domain, that is, you may do anything you want with it.

Download

Latest version is 0.78, released 11 May 2012, and can be found here. It can be built on systems using RedHat Package Manager (rpm) with -tb option to create installable .rpm package. On a Debian GNU/Linux system, the preferred way to install it is to use standard apt repository. For other versions of the package and pre-built rpms look here.

Enjoy. Michael Tokarev, mjt+cdb {at} tls {dot} msk {dot} ru.