DB Berkeley

Table des matières

1 Introduction

A priori, il n'y a pas de page de manuel, il faut utiliser la doc file:///usr/share/doc/db5.3-doc/api_reference/C/index.html

```
# apt-get install libdb-dev db5.3-doc db-util
$ ls /usr/share/doc/db5.3-doc/index.html
```

Makefile

```
CFLAGS=-Wall -O0 -g
LDFLAGS=-ldb

test%: test%.o
gcc $(LDFLAGS) $^ -o $@

test%.o: test%.c
gcc -c $(CFLAGS) $< -o $@

all: test1 test2
clean:
rm -f *.o test?
```

The valgrind errors are expected unless you configure Berkeley DB with the '–enable-umrw' flag, which ensures that all bytes are initialized with zeros, even if they are never read by Berkeley DB (e.g., padding bytes in data structures).

```
==25180== Syscall param pwrite64(buf) points to uninitialised byte(s)
==25180==    at 0x4C13BDC: pwrite64 (pwrite64.c:29)
==25180==    by 0x49AADD8: __os_io (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x499665E: ??? (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x4996B79: __memp_bhwrite (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x49A6BF8: __memp_sync_int (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x49A7393: __memp_fsync (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x493E218: __db_sync (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x493BA85: __db_refresh (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x493C2A1: __db_close (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x494E07C: __db_close_pp (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x10887D: bdbClose (test1.c:87)
==25180==    by 0x108D0F: main (test1.c:237)
==25180==  Address 0x4c28cd8 is 3,944 bytes inside a block of size 4,156 alloc'd
==25180==    at 0x482E27C: malloc (vg_replace_malloc.c:299)
==25180==    by 0x49A8048: __os_malloc (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x497360F: __env_alloc (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x4994748: __memp_alloc (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
```

```
==25180==    by 0x4998544: __memp_fget (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x489B2A4: __bam_get_root (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x489BB48: __bam_search (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x488727C: ??? (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x488B8B4: ??? (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x49428C9: __dbc_iput (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x4943E66: __dbc_put (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==    by 0x493D805: __db_put (in /usr/lib/i386-linux-gnu/libdb-5.3.so)
==25180==  Uninitialised value was created by a stack allocation
==25180==    at 0x108B21: main (test1.c:192)
```

2 Taille fixe

test1.c : enregistrements de taille fixe

```
#include <stdio.h>
#include <string.h> // memset
#include <db.h>

#define TRUE 1
#define FALSE 0

typedef struct MyRecord
{
  char id[10];
  char label1[10];
  char label2[10];
} MyRecord;

/*=========================================================================
 * Function   : bdbOpen
 * Description: open a Berkeley database
 * Synopsis   : int bdbOpen(DB **dbp, char* path)
 * Input      : char* path: path of the database file
 * Output     : DB **dbp: database pointer
 *              TRUE on success
 =========================================================================*/
int bdbOpen(DB **dbp, char* path)
{
  int rc = FALSE;
  u_int32_t flags = 0; /* database open flags */
  int err = 0;         /* function return value */

  if (!dbp) {
    printf("please provide a DB** pointer");
    goto error;
  }

  /* Initialize the structure. This
   * database is not opened in an environment,
   * so the environment pointer is NULL. */
  if ((err = db_create(dbp, NULL, 0))) {
    printf("db_create fails: %s\n", db_strerror(err));
    goto error;
  }

  /* Database open flags */
  flags = DB_CREATE;    /* If the database does not exist,
```

```
 * create it.*/

  /* open the database */
  if ((err = (*dbp)->open(*dbp,        /* DB structure pointer */
  NULL,       /* Transaction pointer */
  path,       /* On-disk file that holds the database. */
  NULL,       /* Optional logical database name */
  DB_BTREE,   /* Database access method */
  flags,      /* Open flags */
  0))) {        /* File mode (using defaults) */
    printf("dbp->open fails: %s\n", db_strerror(err));
    goto error;
  }

  rc = TRUE;
 error:
  if (!rc) {
    printf("bdbOpen fails\n");
  }
  return rc;
}

/*=========================================================================
 * Function   : bdbClose
 * Description: close a Berkeley database
 * Synopsis   : int bdbClose(DB **dbp)
 * Input      : DB **dbp: database pointer
 * Output     : DB **dbp: database pointer is unset to NULL
 *              TRUE on success
 =========================================================================*/
int bdbClose(DB **dbp)
{
  int rc = FALSE;
  int err = 0;

  if (!dbp) {
    printf("please provide a DB** pointer");
    goto error;
  }

  /* When we're done with the database, close it. */
  if (!(*dbp)) goto end;

  if ((err = (*dbp)->close(*dbp, 0))) {
    printf("dbp->open fails: %s\n", db_strerror(err));
    goto error;
  }

  *dbp = 0;
 end:
  rc = TRUE;
 error:
  if (!rc) {
    printf("bdbClose fails\n");
  }
  return rc;
}
```

```
/*************************************/

int addMyRecord(DB *dbp, MyRecord* record)
{
  int rc = FALSE;
  int err = 0;
  DBT key;
  DBT data;

  if (!dbp) {
    printf("please provide a DB* pointer");
    goto error;
  }

  /* Zero out the DBTs before using them. */
  memset(&key, 0, sizeof(DBT));
  memset(&data, 0, sizeof(DBT));

  key.data = record->id;
  key.size = strlen(record->id) + 1;

  data.data = record;
  data.size = sizeof(MyRecord);

  if ((err = dbp->put(dbp, NULL, &key, &data, DB_NOOVERWRITE))) {
    if (err == DB_KEYEXIST) {
      printf("addMyRecord fails: key '%s' already there\n", record->id);
    }
    else {
      printf("sbp->put fails: %s\n", db_strerror(err));
    }
    goto error;
  }

  rc = TRUE;
 error:
  if (!rc) {
    printf("addMyRecord fails\n");
  }
  return rc;
}

int getMyRecord(DB *dbp, char* id, MyRecord* record)
{
  int rc = FALSE;
  int err = 0;
  DBT key;
  DBT data;

  if (!dbp) {
    printf("please provide a DB* pointer");
    goto error;
  }

  memset(record, 0, sizeof(MyRecord));

  /* Zero out the DBTs before using them. */
  memset(&key, 0, sizeof(DBT));
```

```
  memset(&data, 0, sizeof(DBT));

  key.data = id;
  key.size = strlen(id) + 1;

  /* Use our memory to retrieve the structure */
  data.data = record;
  data.ulen = sizeof(MyRecord);
  data.flags = DB_DBT_USERMEM;

  if ((err = dbp->get(dbp, NULL, &key, &data, 0))) {
    if (err == DB_NOTFOUND) {
      printf("getMyRecord fails: no '%s' key there\n", id);
      goto end;
    }
    printf("sbp->put fails: %s\n", db_strerror(err));
    goto error;
  }

 end:
  rc = TRUE;
 error:
  if (!rc) {
    printf("getMyRecord fails\n");
  }
  return rc;
}


int main(int argc, char** argv)
{
  int rc = 0;
  DB *dbp;           /* DB structure handle */
  char id[10] = "";
  MyRecord record;
  int i;

  if (argc < 2) {
    printf("please provide ORDER arcument (get or put)\n");
    goto error;
  }

  if (!bdbOpen(&dbp, "my_db.db")) goto error;

  if (!strcmp(argv[1], "put")) {
    printf("do put\n");

    for (i=0; i<5; ++i) {
      sprintf(record.id, "id_%i", i);
      sprintf(record.label1, "label1_%i", i);
      sprintf(record.label2, "label2_%i", i);

      if (!addMyRecord(dbp, &record)) goto error;
    }
    goto end;
  }

  if (!strcmp(argv[1], "get")) {
```

```
    printf("do get\n");

    for (i=0; i<6; ++i) {
      sprintf(id, "id_%i", i);

      if (!getMyRecord(dbp, id, &record)) continue;
      printf("get %s %s %s\n", record.id, record.label1, record.label2);
    }
    goto end;
  }

  printf("bad ORDER argument '%s', get or put was expected\n", argv[1]);
  goto error;

 end:
  rc = 0;
 error:
  if (!bdbClose(&dbp)) rc = 1;
  printf("%s\n", rc?"failure":"success");
  return rc;
}
```

3 Taille variable

test2.c : enregistrements de taille variable

```
#include <stdio.h>
#include <string.h> // memset
#include <stdlib.h> // malloc
#include <db.h>

#define TRUE 1
#define FALSE 0

typedef struct MyRecord
{
  char* id;
  char* label1;
  char* label2;
} MyRecord;

/*====================================================================
 * Function   : bdbOpen
 * Description: open a Berkeley database
 * Synopsis   : int bdbOpen(DB **dbp, char* path)
 * Input      : char* path: path of the database file
 * Output     : DB **dbp: database pointer
 *              TRUE on success
 ====================================================================*/
int
bdbOpen(DB **dbp, char* path)
{
  int rc = FALSE;
  u_int32_t flags = 0; /* database open flags */
  int err = 0;         /* function return value */

  if (!dbp) {
    printf("please provide a DB** pointer");
```

```
    goto error;
  }

  /* Initialize the structure. This
   * database is not opened in an environment,
   * so the environment pointer is NULL. */
  if ((err = db_create(dbp, NULL, 0))) {
    printf("db_create fails: %s\n", db_strerror(err));
    goto error;
  }

  /* Database open flags */
  flags = DB_CREATE;    /* If the database does not exist,
 * create it.*/

  /* open the database */
  if ((err = (*dbp)->open(*dbp,        /* DB structure pointer */
  NULL,       /* Transaction pointer */
  path,       /* On-disk file that holds the database. */
  NULL,       /* Optional logical database name */
  DB_BTREE,   /* Database access method */
  flags,      /* Open flags */
  0))) {        /* File mode (using defaults) */
    printf("dbp->open fails: %s\n", db_strerror(err));
    goto error;
  }

  rc = TRUE;
 error:
  if (!rc) {
    printf("bdbOpen fails\n");
  }
  return rc;
}

/*====================================================================
 * Function   : bdbClose
 * Description: close a Berkeley database
 * Synopsis   : int bdbClose(DB **dbp)
 * Input      : DB **dbp: database pointer
 * Output     : DB **dbp: database pointer is unset to NULL
 *              TRUE on success
 ====================================================================*/
int
bdbClose(DB **dbp)
{
  int rc = FALSE;
  int err = 0;

  if (!dbp) {
    printf("please provide a DB** pointer");
    goto error;
  }

  /* When we're done with the database, close it. */
  if (!(*dbp)) goto end;

  if ((err = (*dbp)->close(*dbp, 0))) {
```

```
    printf("dbp->open fails: %s\n", db_strerror(err));
    goto error;
  }

  *dbp = 0;
 end:
  rc = TRUE;
 error:
  if (!rc) {
    printf("bdbClose fails\n");
  }
  return rc;
}

/************************************/

char*
packString(char *buffer, char *string)
{
  int len = 0;

  len = strlen(string);
  strncpy(buffer, string, len+1);

  return(buffer + len + 1);
}

char*
unpackString(char *buffer, char **string)
{
  int len = 0;

  *string = buffer;
  len = strlen(*string);

  return(buffer + len + 1);
}

int
addMyRecord(DB *dbp, MyRecord* record)
{
  int rc = FALSE;
  char *buffer = 0;
  char* ptr = 0;
  int len = 0;
  int err = 0;
  DBT key;
  DBT data;

  if (!dbp) {
    printf("please provide a DB* pointer");
    goto error;
  }

  len = strlen(record->id) + strlen(record->label1) + strlen(record->label2) +3;

  if (!(buffer = malloc(len))) {
    printf("malloc fails\n");
```

```
    goto error;
  }

  ptr = packString(buffer, record->id);
  ptr = packString(ptr, record->label1);
  ptr = packString(ptr, record->label2);

  /* Zero out the DBTs before using them. */
  memset(&key, 0, sizeof(DBT));
  memset(&data, 0, sizeof(DBT));

  key.data = record->id;
  key.size = strlen(record->id) + 1;

  data.data = buffer;
  data.size = len;

  if ((err = dbp->put(dbp, NULL, &key, &data, DB_NOOVERWRITE))) {
    if (err == DB_KEYEXIST) {
      printf("addMyRecord fails: key '%s' already there\n", record->id);
    }
    else {
      printf("sbp->put fails: %s\n", db_strerror(err));
    }
    goto error;
  }

  rc = TRUE;
 error:
  if (buffer) free(buffer);
  if (!rc) {
    printf("addMyRecord fails\n");
  }
  return rc;
}

int
getMyRecord(DB *dbp, char* id, MyRecord* record)
{
  int rc = FALSE;
  int err = 0;
  DBT key;
  DBT data;
  //char buffer[128];
  char* ptr = 0;

  if (!dbp) {
    printf("please provide a DB* pointer");
    goto error;
  }

  memset(record, 0, sizeof(MyRecord));

  /* Zero out the DBTs before using them. */
  memset(&key, 0, sizeof(DBT));
  memset(&data, 0, sizeof(DBT));

  key.data = id;
```

```
  key.size = strlen(id) + 1;

  if ((err = dbp->get(dbp, NULL, &key, &data, 0))) {
    if (err == DB_NOTFOUND) {
      printf("getMyRecord fails: no '%s' key there\n", id);
      goto end;
    }
    printf("sbp->put fails: %s\n", db_strerror(err));
    goto error;
  }

  ptr = data.data;
  ptr = unpackString(ptr, &(record->id));
  ptr = unpackString(ptr, &(record->label1));
  ptr = unpackString(ptr, &(record->label2));

 end:
  rc = TRUE;
 error:
  if (!rc) {
    printf("getMyRecord fails\n");
  }
  return rc;
}


int main(int argc, char** argv)
{
  int rc = 0;
  DB *dbp = 0;        /* DB structure handle */
  char id[10] = "";
  char label1[10];
  char label2[10];
  MyRecord record;
  int i;

  if (argc < 2) {
    printf("please provide ORDER arcument (get or put)\n");
    goto error;
  }

  if (!bdbOpen(&dbp, "my_db.db")) goto error;

  if (!strcmp(argv[1], "put")) {
    printf("do put\n");

    record.id = id;
    record.label1 = label1;
    record.label2 = label2;
    for (i=0; i<5; ++i) {
      sprintf(record.id, "id_%i", i);
      sprintf(record.label1, "label1_%i", i);
      sprintf(record.label2, "label2_%i", i);

      if (!addMyRecord(dbp, &record)) goto error;
    }
    goto end;
  }
```

```
  if (!strcmp(argv[1], "get")) {
    printf("do get\n");

    for (i=0; i<6; ++i) {
      sprintf(id, "id_%i", i);

      if (!getMyRecord(dbp, id, &record)) continue;
      printf("get %s %s %s\n", record.id, record.label1, record.label2);
    }
    goto end;
  }

  printf("bad ORDER argument '%s', get or put was expected\n", argv[1]);
  goto error;

 end:
  rc = 0;
 error:
  if (!bdbClose(&dbp)) rc = 1;
  printf("%s\n", rc?"failure":"success");
  return rc;
}
```

4 Curseurs

test3.c : on accepte les doublons sur une même clé.

```
#include <stdio.h>
#include <string.h> // memset
#include <stdlib.h> // malloc
#include <db.h>

#define TRUE 1
#define FALSE 0

typedef struct MyRecord
{
  char* id;
  char* label1;
  char* label2;
} MyRecord;

/*====================================================================
 * Function   : bdbOpen
 * Description: open a Berkeley database
 * Synopsis   : int bdbOpen(DB **dbp, char* path)
 * Input      : char* path: path of the database file
 * Output     : DB **dbp: database pointer
 *              TRUE on success
 ====================================================================*/
int
bdbOpen(DB **dbp, char* path)
{
  int rc = FALSE;
  u_int32_t flags = 0; /* database open flags */
  int err = 0;         /* function return value */
```

```
  if (!dbp) {
    printf("please provide a DB** pointer");
    goto error;
  }

  /* Initialize the structure. This
   * database is not opened in an environment,
   * so the environment pointer is NULL. */
  if ((err = db_create(dbp, NULL, 0))) {
    printf("db_create fails: %s\n", db_strerror(err));
    goto error;
  }

  if ((err = (*dbp)->set_flags(*dbp, DB_DUP))) {
    printf("dbp->set_flags fails: %s\n", db_strerror(err));
    goto error;
  }

  /* Database open flags */
  flags = DB_CREATE;    /* If the database does not exist,
 * create it.*/

  /* open the database */
  if ((err = (*dbp)->open(*dbp,        /* DB structure pointer */
  NULL,       /* Transaction pointer */
  path,       /* On-disk file that holds the database. */
  NULL,       /* Optional logical database name */
  DB_BTREE,   /* Database access method */
  flags,      /* Open flags */
  0))) {        /* File mode (using defaults) */
    printf("dbp->open fails: %s\n", db_strerror(err));
    goto error;
  }

  rc = TRUE;
 error:
  if (!rc) {
    printf("bdbOpen fails\n");
  }
  return rc;
}

/*====================================================================
 * Function   : bdbClose
 * Description: close a Berkeley database
 * Synopsis   : int bdbClose(DB **dbp)
 * Input      : DB **dbp: database pointer
 * Output     : DB **dbp: database pointer is unset to NULL
 *              TRUE on success
 ====================================================================*/
int
bdbClose(DB **dbp)
{
  int rc = FALSE;
  int err = 0;

  if (!dbp) {
    printf("please provide a DB** pointer");
```

```
    goto error;
  }

  /* When we're done with the database, close it. */
  if (!(*dbp)) goto end;

  if ((err = (*dbp)->close(*dbp, 0))) {
    printf("dbp->open fails: %s\n", db_strerror(err));
    goto error;
  }

  *dbp = 0;
 end:
  rc = TRUE;
 error:
  if (!rc) {
    printf("bdbClose fails\n");
  }
  return rc;
}

/*************************************/

char*
packString(char *buffer, char *string)
{
  int len = 0;

  len = strlen(string);
  strncpy(buffer, string, len+1);

  return(buffer + len + 1);
}

char*
unpackString(char *buffer, char **string)
{
  int len = 0;

  *string = buffer;
  len = strlen(*string);

  return(buffer + len + 1);
}

int
addMyRecord(DB *dbp, MyRecord* record)
{
  int rc = FALSE;
  char *buffer = 0;
  char* ptr = 0;
  int len = 0;
  int err = 0;
  DBT key;
  DBT data;

  if (!dbp) {
    printf("please provide a DB* pointer");
```

```
    goto error;
  }

  len = strlen(record->id) + strlen(record->label1) + strlen(record->label2) +3;

  if (!(buffer = malloc(len))) {
    printf("malloc fails\n");
    goto error;
  }

  ptr = packString(buffer, record->id);
  ptr = packString(ptr, record->label1);
  ptr = packString(ptr, record->label2);

  /* Zero out the DBTs before using them. */
  memset(&key, 0, sizeof(DBT));
  memset(&data, 0, sizeof(DBT));

  key.data = record->id;
  key.size = strlen(record->id) + 1;

  data.data = buffer;
  data.size = len;

  if ((err = dbp->put(dbp, NULL, &key, &data, 0/*DB_NOOVERWRITE*/))) {
    if (err == DB_KEYEXIST) {
      printf("addMyRecord fails: key '%s' already there\n", record->id);
    }
    else {
      printf("sbp->put fails: %s\n", db_strerror(err));
    }
    goto error;
  }

  rc = TRUE;
 error:
  if (buffer) free(buffer);
  if (!rc) {
    printf("addMyRecord fails\n");
  }
  return rc;
}

int
getMyRecords(DB *dbp, char* id, MyRecord* record)
{
  int rc = FALSE;
  int err = 0;
  DBC *cursorp = 0;
  DBT key;
  DBT data;
  //char buffer[128];
  char* ptr = 0;

  if (!dbp) {
    printf("please provide a DB* pointer");
    goto error;
  }
```

```
  memset(record, 0, sizeof(MyRecord));

  /* Get a cursor */
  if ((err = dbp->cursor(dbp, NULL, &cursorp, 0))) {
    printf("dbp->cursor fails: %s\n", db_strerror(err));
    goto error;
  }

  /* Zero out the DBTs before using them. */
  memset(&key, 0, sizeof(DBT));
  memset(&data, 0, sizeof(DBT));

  key.data = id;
  key.size = strlen(id) + 1;

  err = cursorp->get(cursorp, &key, &data, DB_SET);
  while (!err && !strcmp(key.data, id)) {
    ptr = data.data;
    ptr = unpackString(ptr, &(record->id));
    ptr = unpackString(ptr, &(record->label1));
    ptr = unpackString(ptr, &(record->label2));

    printf("get %s %s %s\n", record->id, record->label1, record->label2);
    err = cursorp->get(cursorp, &key, &data, DB_NEXT);
  }

  if (err && err != DB_NOTFOUND) {
    printf("cursorp->get fails: %s\n", db_strerror(err));
    goto error;
  }

  if ((err = cursorp->close(cursorp))) {
    printf("cursorp->close fails: %s\n", db_strerror(err));
    goto error;
  }

  rc = TRUE;
 error:
  if (!rc) {
    printf("getMyRecord fails\n");
  }
  return rc;
}

int main(int argc, char** argv)
{
  int rc = 0;
  DB *dbp = 0;        /* DB structure handle */
  char id[10] = "";
  char label1[10];
  char label2[10];
  MyRecord record;
  int i;

  if (argc < 2) {
    printf("please provide ORDER arcument (get or put)\n");
    goto error;
```

```
  }

  if (!bdbOpen(&dbp, "my_db.db")) goto error;

  if (!strcmp(argv[1], "put")) {
    printf("do put\n");

    record.id = id;
    record.label1 = label1;
    record.label2 = label2;
    for (i=0; i<5; ++i) {
      sprintf(record.id, "id_%i", i);
      sprintf(record.label1, "label1_%i", i);
      sprintf(record.label2, "label2_%i", i);

      if (!addMyRecord(dbp, &record)) goto error;
    }
    goto end;
  }

  if (!strcmp(argv[1], "get")) {
    printf("do get\n");

    for (i=0; i<6; ++i) {
      sprintf(id, "id_%i", i);

      printf("-- %s --\n", id);
      if (!getMyRecords(dbp, id, &record)) continue;
    }
    goto end;
  }

  printf("bad ORDER argument '%s', get or put was expected\n", argv[1]);
  goto error;

 end:
  rc = 0;
 error:
  if (!bdbClose(&dbp)) rc = 1;
  printf("%s\n", rc?"failure":"success");
  return rc;
}
```