Oracle Berkeley DB

Porting Guide

Release 4.7

ORACLE®

BERKELEY DB

Legal Notice

Published 4/25/2008

Table of Contents

Preface

The Berkeley DB family of open source, embeddable databases provides developers with fast, reliable persistence with zero administration. Often deployed as "edge" databases, the Berkeley DB family provides very high performance, reliability, scalability, and availability for application use cases that do not require SQL.

As an open source database, Berkeley DB works on many different platforms, from Wind River's Tornado system, to VMS, to Windows NT and Windows 95, and most existing UNIX platforms. It runs on 32 and 64-bit machines, little or big-endian.

Berkeley DB Porting Guide provides the information you need to port Berkeley DB to additional platforms.

Conventions Used in this Book

The following typographical conventions are used within in this manual:

Structure names are represented in `monospaced font`, as are `method names`. For example: "`DB->open()` is a method on a `DB` handle."

Variable or non-literal text is presented in *italics*. For example: "Go to your *DB_INSTALL* directory."

Program examples are displayed in a `monospaced font` on a shaded background. For example:

```
/* File: gettingstarted_common.h */
typedef struct stock_dbs {
    DB *inventory_dbp; /* Database containing inventory information */
    DB *vendor_dbp;    /* Database containing vendor information */

    char *db_home_dir;       /* Directory containing the database files */
    char *inventory_db_name; /* Name of the inventory database */
    char *vendor_db_name;    /* Name of the vendor database */
} STOCK_DBS;
```

Finally, notes of interest are represented using a note block such as this.

Audience

This guide is intended for programmers porting Berkeley DB to a new platform. It assumes that these programmers possess:

- Familiarity with standard ANSI C and POSIX C 1003.1 and 1003.2 library and system calls.

- Working knowledge of the target platform as well as the development tools (for example, compilers, linkers, and debuggers) available on that platform.

For More Information

Beyond this manual, you may also find the following sources of information useful when building a DB application:

- Getting Started with Berkeley DB for C
  [http://www.oracle.com/technology/documentation/berkeley-db/db/gsg/C/BerkeleyDB-Core-C-GSG.pdf]
- Getting Started with Transaction Processing for C
  [http://www.oracle.com/technology/documentation/berkeley-db/db/gsg_txn/C/BerkeleyDB-Core-C-Txn.pdf]
- Berkeley DB Getting Started with Replicated Applications for C
  [http://www.oracle.com/technology/documentation/berkeley-db/db/gsg_db_rep/C/Replication_C_GSG.pdf]
- Berkeley DB Programmer's Reference Guide
  [http://www.oracle.com/technology/documentation/berkeley-db/db/ref/toc.html]
- Berkeley DB C API
  [http://www.oracle.com/technology/documentation/berkeley-db/db/api_c/frame.html]

Chapter 1. Introduction to Porting Berkeley DB

Berkeley DB is an open source database product that supports a variety of platforms. When there is a need to run Berkeley DB on a platform that is currently not supported, DB is distributed in source code form that you can use as base source to port Berkeley DB to that platform.

Before you begin actually porting Berkeley DB, you need an understanding of the:

Types of Berkeley DB ports

There are several types of Berkeley DB ports:

- Ports developed and supported by Oracle
- Ports developed by a customer or a partner, but which Oracle has agreed to support.
- Ports developed, maintained, and supported by a customer or partner.

For a port developed by a customer or a partner, the general steps for porting Berkeley DB to a new platform are the same whether or not Oracle has agreed to support Berkeley DB on the new platform. For example, after you complete the port you send it to Berkeley DB as described in Integrating Changes into the Berkeley DB Source Code (page 9). However, there are some differences.

When Oracle Has Agreed to Support Berkeley DB on the New Platform

When porting Berkeley DB to a platform that Oracle has agreed to support, you need to have Berkeley DB engineering review your port at various points. These review points are discussed more fully in Integrating Changes into the Berkeley DB Source Code (page 9), Modifying the Tests (page 7), and Reviewing the Results of the Tests (page 8).

It is up to you to submit the results of the tests (test_micro, test_os, test_mutex, and, if possible, the entire tcl test suit) for review by Oracle Berkelely DB engineering in order for Oracle to consider providing support for Berkeley DB on a new platform.

You must also assign copyrights for your changes to any part of Berkeley DB to "Oracle Corporation" and attest to the fact that you are not infringing on any software patents for the changes to be included in the general Berekely DB distribution.

Once the port is certified, Oracle provides support for Berkeley DB on the new platform in the same manner that it does for Berkeley DB running on other established platforms.

When Oracle has Not Agreed to Support Berkeley DB on the New Platform

When Oracle has *not* agreed to support Berkeley DB on the new platform, the customer or partner assume the responsibility of front-line support. When it is determined that there is a problem in the code that was not modified by the customer or partner, then Berkeley DB engineering provides support to the customer or vendor who implemented the port, However, in these cases, Oracle needs access to the platform and hardware for diagnosing, debugging, and testing.

Berkeley DB Porting Process

As with any porting project, porting Berkeley DB to a new platform consists of the following process:

1. Determine the modifications that you need to make in the base code and create a working executable of Berkeley DB on the target platform as described in Creating a New Berkeley DB Binary (page 3).

2. Perform final test and quality assurance functions on the target platform as described in Testing and Certifying the Port (page 7).

Chapter 2. Creating a New Berkeley DB Binary

Creating a new Berkeley DB executable on the target platform, involves:

1. Creating a Base Build of Berkeley DB (page 3)
2. Determining the Scope of the Modifications (page 3)
3. Building on the Target Platform (page 6)

Creating a Base Build of Berkeley DB

The simplest way to begin a port is to attempt to configure and build Berkeley DB on a UNIX or UNIX-like system. This gives you a list of the files that you needed to build Berkeley DB as well as the configuration files you can use as a starting point for building on your target port.

To create a base build of Berkeley DB, following the instructions in the *Berkeley DB Programmer's Reference Guide*:

1. Download a Berkeley DB distribution from http://www.oracle.com/technology/software/products/berkeley-db/db/.
2. Build Berkeley DB.

Determining the Scope of the Modifications

Once you have a good build of Berkeley DB on a UNIX or UNIX-like system, look over the code to determine what type of code changes you need to make so that you can successfully build Berkeley DB on your target system. This process involves determining:

- Do Changes Need to be Made to the Operating System Functionality? (page 3)
- Are Some Standard Functions Missing on the Target Platform? (page 5)
- How Will the Port Handle Shared Memory? (page 5)
- What Type of Mutexes Will the Port Use? (page 6)
- Do Any Other Changes Need to be Made? (page 6)

Do Changes Need to be Made to the Operating System Functionality?

Berkeley DB uses about forty operating system primitives. The Berkeley DB distribution contains files which are wrappers around these operating system primitives that act as an abstraction layer to separate the main Berkeley DB code from operating system and

architecture-specific components. You *must* port these files (or versions of these files) whenever you port Berkeley DB to a new platform.

Within a Berkeley DB distribution, typically, there is only a single version of these files for all platforms that Berkeley DB supports. Those versions of the files live in the `os` directory of the distribution and follow the ANSI C and POSIX 1003.1 standards. Within each file, there is usually one, but sometimes several functions (for example, the `os_alloc.c` file contains the `malloc`, `realloc`, `strdup`, and `free` functions). The following table describes the files in the os directory of the Berkeley DB distribution.

Source file	Description
os_abort.c	abort()
os_abs.c	Return if a filename is an absolute path name
os_addrinfo.c	getaddrinfo(), freeaddrinfo()
os_alloc.c	malloc(), realloc(), strdup(), free()
os_clock.c	clock_gettime()
os_config.c	Minor run-time configuration information
os_ctime.c	ctime()
os_dir.c	Return a list of files for a directory
os_errno.c	Library and system error translation
os_fid.c	Return a unique identifier for a file
os_fsync.c	fsync()
os_handle.c	Return a file handle
os_pid.c	Return a unique identifier for a threa
os_map.c	Shared memory mapping
os_mkdir.c	mkdir()
os_oflags.c	open() Used to convert open flags to Berkeley DB flags
os_open.c	Return a file handle
os_rename.c	rename()
os_root.c	Return if application has special permissions
os_rpath.c	Return last separator in a path
os_rw.c	read(), write()
os_seek.c	lseek()
os_sleep.c	sleep()
os_spin.c	Return the number of test-and-set mutex spins before blocking
os_stat.c	stat()
os_tmpdir.c	Return the directory name used by the system for temporary files
os_truncate.c	ftruncate()

Source file	Description
os_uid.c	Return unique 32-bit id
os_unlink.c	unlink()
os.yield.c	yield()

When the operating system primitives on the target platform are identical or close to the POSIX semantics that Berkeley DB requiress, then no code changes or minimal code changes to the files in the `os` directory are required. If the operating system primitives are quite different, then some code changes may be required to bridge the gap between the requirements of Berkeley DB and what the operating system provides.

Where different code is required, you write an entirely different version of the file and place it in an `os_`*xxx* directory where *xxx* represents a platform name. There are `os_`*xxx* subdirectories in the Berkeley DB distribution for several established non-POSIX platforms. For example, there is a he `os_vxworks` directory that contains VxWorks versions of some of the files in the os directory, and Windows versions of some files are in the `os_windows` directory. If your target platform needs a different version of a file, you will need to write that file and place it in a new `os_`*xxx* directory that you create for your target platform.

Are Some Standard Functions Missing on the Target Platform?

In some cases, the target platform may not provide the few POSIX functions required by Berkeley DB or the functions provided by the target platform may not operate in a standard compliant way. Berkeley DB provides replacement functionsin the `clib` directory of the Berkeley DB distribution.

You need to determine how your target platfrom handles these functions:

- When the target platform does *not* have a POSIX function required by Berkeley DB, no action is required on your part. When Berekely DB cannot find one of these functions on the target platform, it automatically uses the replacement functions supplied in the `clib` directory of the Berkeley DB distribution. For example, if the target platform does not have the `atoi` or `strtol` functions, Berkeley DB uses `clib/atoi.c` and `clib/strtol.c`.

- When the target platform has a function required by Berekely DB, but that function operates in a non-standard compliant way, you can code to the replacement functions supplied in the `clib` directory.

How Will the Port Handle Shared Memory?

In order to write multiprocess database applications (not multithreaded, but threads of control running in different address spaces), Berkeley DB must be able to name pieces of shared memory and access them from multiple processes.

On UNIX/POSIX systems, Berkeley DB uses `mmap` and `shmget` for that purpose, but any interface that provides access to named shared memory is sufficient. If you have a simple, flat address space, you should be able to use the code in `os_vxworks/os_map.c` as a starting point for the port.

If you are not intending to write multiprocess database applications, then this won't be necessary, as Berkeley DB can simply allocate memory from the heap if all threads of control will live in a single address space.

What Type of Mutexes Will the Port Use?

Berkeley DB requires some form of self-blocking mutual exclusion mutex. Blocking mutexes are preferred as they tend to be less CPU-expensive and less likely to cause thrashing. If blocking mutexes are not available, however, test-and-set will work as well. The code for mutexes is in two places in the system: the include file `dbinc/mutex_int.h`, and the distribution directory `mutex`.

Do Any Other Changes Need to be Made?

In most cases, you do not need to make any changes to the Berkeley DB source code that is not in the abstraction layer (that is, that is in the `os` directory) as that code is designed to be platform-independent code. However, in some situations, the compiler for the target platform is non-standard and may raise errors when compiling some aspects of the Berkeley DB code (for example, additional casting may be required, or a certain type may cause a problem). In these cases, you will need to modify the generic Berkeley DB code in order to have error-free compilation.

Building on the Target Platform

Once you have an idea of the scope of the modifications, use the files generated on the UNIX system to help you create any compiler or linker tool chain files that you need to build on the target platform. At this point, start building the Berkeley DB on the target platform making the changes to the code that you identified earlier in the process. Once you have identified the modifications that you need to make, change the code accordingly.

Chapter 3. Testing and Certifying the Port

There are several different types of tests available for validating your port of Berkeley DB as discussed in Types of Tests for Berkeley DB (page 7). Testing your port involves:

- Modifying the Tests (page 7)
- Running the Tests (page 8)
- Reviewing the Results of the Tests (page 8)
- Integrating Changes into the Berkeley DB Source Code (page 9)
- Certifying a Port of Berkeley DB (page 9)

Types of Tests for Berkeley DB

There are two types of tests available for testing your port of Berkeley DB:

- The C Tests for Berkeley DB

  There are three types of C tests for Berkeley DB. Each of these is in its own directory:

  - `test_os` contains files that test the system primitives. See the `test_os/Test_OS_Technical_Detail` file for detailed information on these tests (for example, how each function is tested).
  - `test_mutex` contains files that test the use of mutexes in Berkeley DB.
  - `test_micro` contains the C tests that exercise the most common code paths, but it is not intended to be an exhaustive Test Suite. Additionally, it tests the different versions of Berkeley DB (including the new port) against each other. The `test_micro` tests can either be run in a shell or as simple C tests.

- The Berkeley DB Test Suite

  The `test` directory contains the Berkeley DB Test Suite that tests all of the code in Berkeley DB. Using the Test Suite involves using Tool Command Language (Tcl) version 8.4 or later. Running the standard version of the Test Suite executes tests the major functionality of Berkeley DB. A more exhaustive version of the Test Suite runs all the tests several more times, testing encryption, replication, and different page sizes.

Modifying the Tests

There should be no need to make modifications to the tests. However, in a few situations, the test hardware may have some constraints (for example, small amount of memory) which may cause certain tests (which expect more memory) to fail. In these rare situations, you may need to modify the tests to work within the constraints of the platform.

When Oracle has agreed to support Berkeley DB on the new platform, submit any proposed test changes for review and approval by Oracle Engineering before running the tests.

Running the Tests

You test your new port of Berkeley DB by running the tests in the following order:

1. Run the C tests in the following order:

    a. Tests for the system primitives located in the `test_os` directory. To run the tests, follow the instructions in the `test_os/readme` file.

    b. Tests for mutexes located in the `test_mutex` directory. To run the tests, follow the instructions in the `test_mutex/readme` file.

    c. Tests for the common code paths located in the `test_micro` directory. To run the tests in a shell script, follow the instructions in the `test_micro/readme` file. To run the tests as simple C tests, follow the instructions in the `test_micro/readme_embedded` file.

2. If the target platform supports the use of Tcl (version 8.4 or later), run the Test Suite. How you run the Test Suite varies depending on the target platform:

    - If the target platform supports a UNIX-like version of Tcl, then set up Tcl and build the Test Suite as described in "Running the Test Suite under UNIX" in *Berkeley DB Programmer's Reference Guide* at http://www.oracle.com/technology/documentation/berkeley-db/db/ref/build_unix/test.html and, then, run the Test Suite as described in "Running the Test Suite" in *Berkeley DB Programmer's Reference Guide* at http://www.oracle.com/technology/documentation/berkeley-db/db/ref/test/run.html.

    - If the target platform supports a Windows-like version of Tcl, then setup Tcl, and build and run the Test Suite as described in "Running the Test Suite under Windows" in *Berkeley DB Programmer's Reference Guide* at http://www.oracle.com/technology/documentation/berkeley-db/db/ref/build_win/test.html

Reviewing the Results of the Tests

It is up to you to submit the results of the tests (`test_micro`, `test_os`, `test_mutex`, and, if possible, the entire tcl test suit) for review by Oracle Berkelely DB engineering in order for Oracle to consider providing support for Berkeley DB on a new platform.

When Oracle has *not* agreed to support Berkeley DB on the new platform, you are responsible for ensuring that the tests run successfully.

Integrating Changes into the Berkeley DB Source Code

Once you have have a completed your port send all code changes that you made as "diff" files to Berkeley DB development for review.

Exactly how you integrate changes into the Berkeley DB source code varies depending on whether or not Oracle has agreed to support Berkeley DB on the new platform:

At this point, when Oracle has agreed to support Berkeley DB on the new platform:

1. Berkeley DB development reviews the changes, makes the changes (with modifications if necessary) in the source code, creates another snapshot which includes the new platform-specific changes, and sends you the new snapshot.

2. On receipt of the new snapshot, recompile this new snapshot. If the new snapshot compiles correctly without any changes, test the port again.

Certifying a Port of Berkeley DB

When the target platform supports using Tcl, the port is considered certified after a successful standard run (`run_std`) of the Test Suite .

When the target platform does not support using Tcl, a port is considered certified if you see successful message reports for each of the tests in `test_os`, `test_micro`, and `test_mutex`.

Additionally, the configuration and compilation of Berkeley DB must be complete without errors or warninigs of any kind. You must provide specific information about the hardware, software, and compiler used during the porting process, especially versions of all thrid party tools used. If you have other diagnostic tools available, such as memory allocation checking tools, please conduct tests using them as well and provide Oracle engineering with the results. Finally, do not constrain your testing to one configuration. There are many different ways to configure Berkeley DB (that is many options to the "configure" script), please configure, built, and test Berkeley DB on the target platform with as many combinations of these flags as possible to ensure that nothing is missed.