NAME

db_thread − using threads in the DB library.

DESCRIPTION

The DB library is a family of groups of functions that provides a modular programming interface to transactions and record-oriented file access. The library includes support for transactions, locking, logging and file page caching, as well as various indexed access methods. Many of the functional groups (e.g., the file page caching functions) are useful independent of the other DB functions, although some functional groups are explicitly based on other functional groups (e.g., transactions and logging). For a general description of the DB package, see *db_intro*(3).

This manual pages describes the specific details of using DB from within threaded programs.

The DB library is not itself multi-threaded. The library was deliberately architected to not use threads internally because of the portability problems that using threads within the library would introduce. Object handles returned from DB library functions are free-threaded, i.e., threads may use handles concurrently, by specifying the DB_THREAD flag to *db_appinit*(3) (or, in the C++ API, *DbEnv::appinit*(3)) and the other subsystem open functions. Threading is assumed in the Java API, so no special flags are required, and DB functions will always behave as if the DB_THREAD flag was specified.

DB supports multi-threaded applications with the caveat that it loads and calls functions that are commonly available in C language environments and which may not themselves be thread-safe. Other than this usage, DB has no static data and maintains no local context between calls to DB functions. To ensure that applications can safely use threads in the context of DB, porters to new operating systems and/or C libraries must confirm that the system and C library functions used by the DB library are thread-safe.

There are some additional caveats about using threads to access the DB library:

- The DB_THREAD flag must be specified for all subsystems either explicitly or via the *db_appinit* (*DbEnv::appinit*) function in the C and C++ APIs. Threading is assumed in the Java API, so no special flags are required, and DB functions will always behave as if the DB_THREAD flag was specified.

  Setting the DB_THREAD flag inconsistently may result in database corruption.

- Only a single thread may call the close function for a returned database or subsystem handle. See *db_open*(3) (*Db::open*(3)) and the appropriate subsystem manual pages for more information.

- Either the DB_DBT_MALLOC or DB_DBT_USERMEM flags must be set in a DBT used for key or data retrieval. See *db_dbt*(3) (*Dbt*(3)) for more information.

- The DB_CURRENT, DB_NEXT and DB_PREV flags to the *log_get* (*DbLog::get*) function may not be used by a free-threaded handle. If such calls are necessary, a thread should explicitly create a unique DB_LOG handle by calling *log_open*(3) (*DbLog::open*). See *db_log*(3) (*DbLog*(3)) for more information.

- Each database operation (i.e., any call to a function underlying the handles returned by *db_open*(3) and *db_cursor*(3)) (*Db*(3) and *Dbc*(3)) is normally performed on behalf of a unique locker. If, within a single thread of control, multiple calls on behalf of the same locker are desired, then transactions must be used. For example, consider the case where a cursor scan locates a record, and then based on that record, accesses some other item in the database. If these are done using the default lockers for the handle, there is no guarantee that these two operations will not conflict. If the application wishes to guarantee that the operations do not conflict, locks must be obtained on behalf of a transaction, instead of the default locker id, and a transaction must be specified to the cursor creation and the subsequent db call.

- Transactions may not span threads, i.e., each transaction must begin and end in the same thread, and each transaction may only be used by a single thread.

- Spinlocks must have been implemented for the compiler/architecture combination. Attempting to specify the DB_THREAD flag will fail if spinlocks are not available.

- The DB library makes a system call to pause for some number of microseconds when it is necessary to wait on a lock. This may not be optimal, especially in a thread-only environment where it will be more efficient to explicitly yield the processor to another thread. It is possible to specify a yield function on an per-application basis, see *db_jump_set*(3) for more information.

COMPILING THREADED APPLICATIONS

Special compile-time flags are required when compiling threaded applications with the UNIX include files on some architectures.

On IRIX, if you are compiling a threaded application, you must compile with the -D_SGI_MP_SOURCE flag:

```
cc -D_SGI_MP_SOURCE ...
```

On OSF/1, if you are compiling a threaded application, you must compile with the -D_REENTRANT flag:

```
cc -D_REENTRANT ...
```

On Solaris, if you are compiling a threaded application, you must compile with the -D_REENTRANT flag and link with the -lthread library:

```
cc -D_REENTRANT ... -lthread
```

SEE ALSO

db_archive(1), *db_checkpoint*(1), *db_deadlock*(1), *db_dump*(1), *db_load*(1), *db_recover*(1), *db_stat*(1), *db_intro*(3), *db_appinit*(3), *db_cursor*(3), *db_dbm*(3), *db_internal*(3), *db_lock*(3), *db_log*(3), *db_mpool*(3), *db_open*(3), *db_thread*(3), *db_txn*(3)