NAME

db_jump_set, db_value_set – replace underlying DB functionality

SYNOPSIS

#include <db.h>

int
db_jump_set(void *func, int which);

int
db_value_set(int value, int which);

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 page describes interfaces to tune or replace underlying system functionality used by the DB library. Few applications should have any need for these interfaces.

db_jump_set

The *db_jump_set* function enables applications to replace underlying DB library functionality by replacing entries in a function call jump table. The *which* argument specifies the entry to be replaced by the argument *func*.

The following values of *which* are supported:

DB_FUNC_CLOSE

Replace all DB calls to the IEEE Std 1003.1b-1993 (‘‘POSIX’’) *close* function with *func*, which must conform to the standard interface.

DB_FUNC_DIRFREE

The DB library requires the ability to return any memory allocated as part of the routine which reads through a directory and creates a list of files that that the directory contains (see DB_FUNC_DIRLIST). The *func* argument must conform to the following interface:

```
int dirfree(char **namesp, int cnt);
```

The *namesp* and *cnt* arguments are the same values as were returned by the DB_FUNC_DIRLIST function.

The *dirfree* function returns the value of *errno* on failure and 0 on success.

DB_FUNC_DIRLIST

The DB library requires the ability to read through a directory and create a list of files that that the directory contains. The *func* argument must conform to the following interface:

```
int dirlist(const char *dir,
        char ***namesp, int *cntp);
```

The *dir* argument is the name of the directory to be searched. The function must return a pointer to an array of nul-terminated file names in the memory location referenced by the argument *namesp*, and a count of the number of elements in the array in the memory location referenced by *cntp*.

The *dirlist* function returns the value of *errno* on failure and 0 on success.

DB_FUNC_EXISTS

The DB library requires the ability to determine if a file exists, and optionally, if it is a file of type directory. The *func* argument must conform to the following interface:

```
int exists(const char *path, int *isdirp);
```

The *path* argument is the pathname of the file to be checked.

If the *isdirp* argument is non-NULL, it must be set to non-0 if *path* is a directory, and 0 if *path* is not a directory.

The *exists* function returns the value of *errno* on failure and 0 on success.

DB_FUNC_FREE
: Replace all DB calls to the ANSI C X3.159-1989 (“ANSI C”) standard *free* function with *func*, which must conform to the standard interface.

DB_FUNC_FSYNC
: Replace all DB calls to the IEEE Std 1003.1b-1993 (“POSIX”) *fsync* function with *func*, which must conform to the standard interface.

DB_FUNC_IOINFO
: The DB library requires the ability to determine the size and I/O characteristics of a file. The *func* argument must conform to the following interface:

```
int ioinfo(const char *path, int fd,
        u_int32_t *mbytesp, u_int32_t *bytesp,
        u_int32_t *iosizep);
```

The *path* argument is the pathname of the file to be checked, and the *fd* argument is an open file descriptor on the file.

If the *mbytesp* and *bytesp* arguments are non-NULL, the *ioinfo* function must return in them the size of the file: the number of megabytes in the file into the memory location referenced by the *mbytesp* argument, and the number of bytes over and above that number of megabytes into the memory location referenced by the *bytesp* argument.

In addition, if the *iosizep* argument is non-NULL, the *ioinfo* function must return the optimum granularity for I/O operations to the file in the memory location referenced by it.

The *ioinfo* function returns the value of *errno* on failure and 0 on success.

DB_FUNC_MALLOC
: Replace all DB calls to the ANSI C X3.159-1989 (“ANSI C”) standard *malloc* function with *func*, which must conform to the standard interface.

DB_FUNC_MAP
: The DB library requires the ability to map a file into memory and to create shared memory regions (which may or may not be backed by files). The *func* argument must conform to the following interface:

```
int map(char *path, int fd, size_t len,
        int is_region, int is_anonymous, int is_rdonly,
        void **addr);
```

The *path* argument is the name of a file. The *fd* argument is an open file descriptor on that file.

The *is_region* argument will be zero if the intention is to map a file into shared memory. In this case, the *map* function must map the first *len* bytes of the file into memory and return a pointer to the mapped location in the memory location referenced by the argument *addr*. In this case, the *is_anonymous* argument will always be zero. The *is_rdonly* argument will be non-zero if the file is considered read-only by the caller.

The *is_region* argument will be non-zero if the memory is intended to be used as a shared memory region for synchronization between DB threads/processes. In this case, the returned memory may be of any kind (e.g., anonymous), but must be able to support semaphores. If the application has previously specified that regions are to be instantiated in anonymous memory (see DB_REGION_ANON below), or the region is being joined and is believed to have been allocated in anonymous shared memory, the *is_anonymous* argument will be non-zero. Additionally, the *path* and *fd* arguments may be ignored (although future *map* calls using the same *path* must return the same memory), and the *is_rdonly* argument will always be zero.

By default, on UNIX systems, the DB library will use the IEEE Std 1003.1b-1993 (‘‘POSIX’’) *mmap*(2) interface to both map regular files into shared memory and create shared memory regions. If the application specifies that shared memory regions be instantiated in anonymous memory (see DB_REGION_ANON below), the *shmget*(2) shared memory segment interface will be used, where available, and the MAP_ANON or MAP_ANONYMOUS options to *mmap*(2) when *shmget*(2) is not available.

When using *shmget*(2), shared memory regions are named, and so multiple processes may share them. When using the *mmap* MAP_ANON or MAP_ANONYMOUS options, shared memory regions are **not** named, and so may only be accessed by a single process and its threads.

HP/UX note:
The *shmget*(2) interfaces are not used on HP/UX, even though they exist, as anonymous memory allocated using *shmget*(2) cannot be used to store semaphores.

The *map* function returns the value of *errno* on failure and 0 on success.

DB_FUNC_OPEN
Replace all DB calls to the IEEE Std 1003.1b-1993 (‘‘POSIX’’) *open* function with *func*, which must conform to the standard interface.

DB_FUNC_READ
Replace all DB calls to the IEEE Std 1003.1b-1993 (‘‘POSIX’’) *read* function with *func*, which must conform to the standard interface.

DB_FUNC_REALLOC
Replace all DB calls to the ANSI C X3.159-1989 (‘‘ANSI C’’) standard *realloc* function with *func*, which must conform to the standard interface.

DB_FUNC_RUNLINK
The DB library requires the ability to remove shared memory regions from the system, whether or not they are backed by regular files. The *func* argument must conform to the following interface:

```
int runlink(char *path);
```

The *path* argument is the path argument specified to the DB_FUNC_MAP function when the region was mapped into memory.

The *runlink* function returns the value of *errno* on failure and 0 on success.

DB_FUNC_SEEK
The DB library requires the ability to specify that a subsequent read from or write to a file will occur at a specific location in that file. The *func* argument must conform to the following interface:

```
int seek(int fd, size_t pgsize, db_pgno_t pageno,
        u_int32_t relative, int rewind, int whence);
```

The *fd* argument is an open file descriptor on the file. The *seek* function must cause a subsequent read from or write to the file to occur at a byte offset specified by the calculation:

(pgsize * pageno) + relative

If *rewind* is non-zero, the byte offset is treated as a backwards seek, not a forwards one.

The *whence* argument specifies where in the file the byte offset is relative to, as described by the IEEE Std 1003.1b-1993 (“POSIX”) *lseek* system call.

The *seek* function returns the value of *errno* on failure and 0 on success.

DB_FUNC_SLEEP
: The DB library requires the ability to cause a process to suspend itself for a period of time, relinquishing control of the processor to any other waiting thread of control. The *func* argument must conform to the following interface:

int sleep(u_long seconds, u_long microseconds);

The *seconds* and *microseconds* arguments specify the amount of time to wait until the suspending thread of control should run again.

The *seconds* and *microseconds* arguments may not be normalized when the *sleep* function is called, i.e., the *microseconds* argument may be greater than 1000000.

The *sleep* function returns the value of *errno* on failure and 0 on success.

DB_FUNC_UNLINK
: Replace all DB calls to the IEEE Std 1003.1b-1993 (“POSIX”) *unlink* function with *func*, which must conform to the standard interface.

DB_FUNC_UNMAP
: The DB library requires the ability to unmap a file or shared memory region from memory. The *func* argument must conform to the following interface:

int unmap(void *addr, size_t len);

The *addr* argument is the argument returned by the DB_FUNC_MAP function when the file or region was mapped into memory, and the *len* argument is the same as the *len* argument specified to the DB_FUNC_MAP function when the file or region was mapped into memory.

The *unmap* function returns the value of *errno* on failure and 0 on success.

DB_FUNC_WRITE
: Replace all DB calls to the IEEE Std 1003.1b-1993 (“POSIX”) *write* function with *func*, which must conform to the standard interface.

DB_FUNC_YIELD
: The DB library requires the ability to yield the processor from the current thread of control to any other waiting threads of control. The *func* argument must conform to the following interface:

int yield(void);

The *yield* function must be able to cause the rescheduling all participants in the current DB environment, whether threaded or not. It may be incorrect to supply a thread *yield* function if more than a single process is operating in the DB environment. This is because many thread-yield functions will not allow other processes to run, and the contested lock may be held by another process, not by another thread.

If no *yield* function is specified, or if the *yield* function returns an error, the function specified by the DB_FUNC_SLEEP entry will be used instead or subsequently, i.e., if no *yield* function is specified, or it is possible for the *yield* function to fail, the *sleep* function **must** cause the processor to reschedule any waiting threads of control for execution.

The *yield* function returns the value of *errno* on failure and 0 on success.

The *db_jump_set* function returns the value of *errno* on failure and 0 on success.

db_value_set

The *db_value_set* function enables applications to specify underlying DB library functionality. The *which* argument specifies the information being set by the argument *value*.

The following values of *which* are supported:

DB_MUTEXLOCKS

Grant all requested mutual exclusion mutexes without testing for their availability. This flag should never be used for any other purpose than debugging.

DB_REGION_ANON

Setting *value* to a non-zero value specifies that shared memory regions are to be created in anonymous memory, and not backed by a regular file. DB_REGION_NAME differs from DB_REGION_ANON in that the former will fail if the shared memory regions cannot be named, that is, if multiple processes cannot use them. See DB_FUNC_MAP for more information.

DB_REGION_INIT

In some applications, the expense of page-faulting the shared memory regions can affect performance, e.g., when the page-fault occurs while holding a lock, other lock requests can convoy and overall throughput may decrease. Setting *value* to a non-zero value specifies that one byte be read from each 4K page of the shared memory region when the region is initialized.

DB_REGION_NAME

Setting *value* to a non-zero value specifies that shared memory regions are to be created in anonymous memory, and not backed by a regular file. DB_REGION_NAME differs from DB_REGION_ANON in that the former will fail if the shared memory regions cannot be named, that is, if multiple processes cannot use them. See DB_FUNC_MAP for more information.

DB_TSL_SPINS

Specify the number of times mutexes should spin without blocking.

This value defaults to 1 on uniprocessor systems and to 50 times the number of processors on multiprocessor systems.

The *db_value_set* function returns the value of *errno* on failure and 0 on success.

ERRORS

The *db_jump_set* function may fail and return *errno* for the following conditions:

[EINVAL]

An invalid flag value or parameter was specified.

BUGS

No type checking is done of the *func* argument, and specifying an invalid replacement routine will cause unpredictable results.

Applications should be careful to replace related functions as a group and at the same time. Replacing DB_FUNC_MALLOC without replacing DB_FUNC_REALLOC is likely to result in unpredictable results.

On Windows/95, files that are opened by multiple processes do not share data correctly. To tell Berkeley DB to use a named region of the paging file to share memory instead, use:

```
db_value_set(1, DB_REGION_NAME);
```

You do not need to do this if your application can guarantee that only one process will be accessing DB files.

On Windows/NT, sharing of data between processes through the paging file does not work correctly, so you should not call *db_value_set*. That will allow DB to use the file itself for sharing, which works correctly.

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)