**NAME**

mpool − shared memory buffer pool

**SYNOPSIS**

```
#include <db.h>
#include <mpool.h>

MPOOL *
mpool_open (DBT *key, int fd, pgno_t pagesize, pgno_t maxcache);

void
mpool_filter (MPOOL *mp, void (*pgin)(void *, pgno_t, void *),
          void (*pgout)(void *, pgno_t, void *), void *pgcookie);

void *
mpool_new (MPOOL *mp, pgno_t *pgnoaddr);

void *
mpool_get (MPOOL *mp, pgno_t pgno, u_int flags);

int
mpool_put (MPOOL *mp, void *pgaddr, u_int flags);

int
mpool_sync (MPOOL *mp);

int
mpool_close (MPOOL *mp);
```

**DESCRIPTION**

*Mpool* is the library interface intended to provide page oriented buffer management of files. The buffers may be shared between processes.

The function *mpool_open* initializes a memory pool. The *key* argument is the byte string used to negotiate between multiple processes wishing to share buffers. If the file buffers are mapped in shared memory, all processes using the same key will share the buffers. If *key* is NULL, the buffers are mapped into private memory. The *fd* argument is a file descriptor for the underlying file, which must be seekable. If *key* is non-NULL and matches a file already being mapped, the *fd* argument is ignored.

The *pagesize* argument is the size, in bytes, of the pages into which the file is broken up. The *maxcache* argument is the maximum number of pages from the underlying file to cache at any one time. This value is not relative to the number of processes which share a file's buffers, but will be the largest value specified by any of the processes sharing the file.

The *mpool_filter* function is intended to make transparent input and output processing of the pages possible. If the *pgin* function is specified, it is called each time a buffer is read into the memory pool from the backing file. If the *pgout* function is specified, it is called each time a buffer is written into the backing file. Both functions are are called with the *pgcookie* pointer, the page number and a pointer to the page to being read or written.

The function *mpool_new* takes an MPOOL pointer and an address as arguments. If a new page can be allocated, a pointer to the page is returned and the page number is stored into the *pgnoaddr* address. Otherwise, NULL is returned and errno is set.

The function *mpool_get* takes a MPOOL pointer and a page number as arguments. If the page exists, a pointer to the page is returned. Otherwise, NULL is returned and errno is set. The flags parameter is not currently used.

The function *mpool_put* unpins the page referenced by *pgaddr*. *Pgaddr* must be an address previously returned by *mpool_get* or *mpool_new*. The flag value is specified by *or*'ing any of the following values:

MPOOL_DIRTY
The page has been modified and needs to be written to the backing file.

*Mpool_put* returns 0 on success and -1 if an error occurs.

The function *mpool_sync* writes all modified pages associated with the MPOOL pointer to the backing file. *Mpool_sync* returns 0 on success and -1 if an error occurs.

The *mpool_close* function free's up any allocated memory associated with the memory pool cookie. Modified pages are **not** written to the backing file. *Mpool_close* returns 0 on success and -1 if an error occurs.

**ERRORS**

The *mpool_open* function may fail and set *errno* for any of the errors specified for the library routine *malloc*(3).

The *mpool_get* function may fail and set *errno* for the following:

[EINVAL] The requested record doesn't exist.

The *mpool_new* and *mpool_get* functions may fail and set *errno* for any of the errors specified for the library routines *read*(2)*,* *write*(2)*,* and *malloc*(3).

The *mpool_sync* function may fail and set *errno* for any of the errors specified for the library routine *write*(2).

The *mpool_close* function may fail and set *errno* for any of the errors specified for the library routine *free*(3).

**SEE ALSO**

*dbopen*(3), *btree*(3), *hash*(3), *recno*(3)