NAME

libpmem − persistent memory support library

SYNOPSIS

```
#include <libpmem.h>
cc ... -lpmem
```

Library API versioning:

```
const char *pmem_check_version(
    unsigned major_required,
    unsigned minor_required);
```

Error handling:

```
const char *pmem_errormsg(void);
```

Other library functions:

A description of other **libpmem** functions can be found on the following manual pages:

- most commonly used functions: **pmem_is_pmem**(3)
- partial flushing operations: **pmem_flush**(3)
- copying to persistent memory: **pmem_memmove_persist**(3)

DESCRIPTION

libpmem provides low−level *persistent memory* (pmem) support for applications using direct access storage (DAX), which is storage that supports load/store access without paging blocks from a block storage device. Some types of *non−volatile memory DIMMs* (NVDIMMs) provide this type of byte addressable access to storage. A *persistent memory aware file system* is typically used to expose the direct access to applications. Memory mapping a file from this type of file system results in the load/store, non−paged access to pmem.

This library is for applications that use persistent memory directly, without the help of any library−supplied transactions or memory allocation. Higher−level libraries that build on **libpmem** are available and are recommended for most applications, see:

- **libpmemobj**(7), a general use persistent memory API, providing memory allocation and transactional operations on variable−sized objects.
- **libpmemblk**(7), providing pmem−resident arrays of fixed−sized blocks with atomic updates.
- **libpmemlog**(7), providing a pmem−resident log file.

Under normal usage, **libpmem** will never print messages or intentionally cause the process to exit. The only exception to this is the debugging information, when enabled, as described under **DEBUGGING AND ERROR HANDLING** below.

CAVEATS

libpmem relies on the library destructor being called from the main thread. For this reason, all functions that might trigger destruction (e.g. **dlclose**(3)) should be called in the main thread. Otherwise some of the resources associated with that thread might not be cleaned up properly.

LIBRARY API VERSIONING

This section describes how the library API is versioned, allowing applications to work with an evolving API.

The **pmem_check_version**() function is used to determine whether the installed **libpmem** supports the version of the library API required by an application. The easiest way to do this is for the application to supply the compile−time version information, supplied by defines in **<libpmem.h>**, like this:

```
reason = pmem_check_version(PMEM_MAJOR_VERSION,
                            PMEM_MINOR_VERSION);
if (reason != NULL) {
    /* version check failed, reason string tells you why */
}
```

Any mismatch in the major version number is considered a failure, but a library with a newer minor version number will pass this check since increasing minor versions imply backwards compatibility.

An application can also check specifically for the existence of an interface by checking for the version where that interface was introduced. These versions are documented in this man page as follows: unless otherwise specified, all interfaces described here are available in version 1.0 of the library. Interfaces added after version 1.0 will contain the text *introduced in version x.y* in the section of this manual describing the feature.

When the version check performed by **pmem_check_version**() is successful, the return value is NULL. Otherwise the return value is a static string describing the reason for failing the version check. The string returned by **pmem_check_version**() must not be modified or freed.

ENVIRONMENT

libpmem can change its default behavior based on the following environment variables. These are largely intended for testing and are not normally required.

- **PMEM_IS_PMEM_FORCE**=*val*

If *val* is 0 (zero), then **pmem_is_pmem**(3) will always return false. Setting *val* to 1 causes **pmem_is_pmem**(3) to always return true. This variable is mostly used for testing but can be used to force pmem behavior on a system where a range of pmem is not detectable as pmem for some reason.

> NOTE: Unlike the other variables, the value of **PMEM_IS_PMEM_FORCE** is not queried (and cached) at library initialization time, but on the first call to **pmem_is_pmem**(3). This means that in case of **libpmemlog**(7), **libpmemblk**(7), and **libpmemobj**(7), **PMEM_IS_PMEM_FORCE** may still be set or modified by the program until the first attempt to create or open the persistent memory pool.

- **PMEM_NO_CLWB**=1

Setting this environment variable to 1 forces **libpmem** to never issue the **CLWB** instruction on Intel hardware, falling back to other cache flush instructions instead (**CLFLUSHOPT** or **CLFLUSH** on Intel hardware). Without this environment variable, **libpmem** will always use the **CLWB** instruction for flushing processor caches on platforms that support the instruction. This variable is intended for use during library testing but may be required for some rare cases where using **CLWB** has a negative impact on performance.

- **PMEM_NO_CLFLUSHOPT**=1

Setting this environment variable to 1 forces **libpmem** to never issue the **CLFLUSHOPT** instruction on Intel hardware, falling back to the **CLFLUSH** instructions instead. Without this environment variable, **libpmem** will always use the **CLFLUSHOPT** instruction for flushing processor caches on platforms that support the instruction, but where **CLWB** is not available. This variable is intended for use during library testing.

- **PMEM_NO_FLUSH**=1

Setting this environment variable to 1 forces most **libpmem** functions to never issue any of **CLFLUSH**, **CLFLUSHOPT** or **CLWB** instructions on Intel hardware. The only exceptions are **pmem_deep_flush**(3) and **pmem_deep_persist**(3) functions.

- **PMEM_NO_FLUSH**=0

Setting this environment variable to 0 forces to always flush CPU caches using one of **CLFLUSH**, **CLFLUSHOPT** or **CLWB** instructions even if **pmem_has_auto_flush**(3) function returns true and the platform supports flushing the processor caches on power loss or system crash.

- **PMEM_NO_MOVNT**=1

Setting this environment variable to 1 forces **libpmem** to never use the *non−temporal* move instructions on Intel hardware. Without this environment variable, **libpmem** will use the non−temporal instructions for copying larger ranges to persistent memory on platforms that support the instructions. This variable is intended for use during library testing.

- **PMEM_MOVNT_THRESHOLD=***val*

This environment variable allows overriding the minimum length of the **pmem_memmove_persist**(3) operations, for which **libpmem** uses *non−temporal* move instructions. Setting this environment variable to 0 forces **libpmem** to always use the *non−temporal* move instructions if available. It has no effect if **PMEM_NO_MOVNT** is set to 1. This variable is intended for use during library testing.

- **PMEM_MMAP_HINT=***val*

This environment variable allows overriding the hint address used by **pmem_map_file**(). If set, it also disables mapping address randomization. This variable is intended for use during library testing and debugging. Setting it to some fairly large value (i.e. 0x10000000000) will very likely result in mapping the file at the specified address (if not used) or at the first unused region above given address, without adding any random offset. When debugging, this makes it easier to calculate the actual address of the persistent memory block, based on its offset in the file. In case of **libpmemobj** it simplifies conversion of a persistent object identifier (OID) into a direct pointer to the object.

> NOTE: **Setting this environment variable affects all the PMDK libraries,** disabling mapping address randomization and causing the specified address to be used as a hint about where to place the mapping.

DEBUGGING AND ERROR HANDLING

If an error is detected during the call to a **libpmem** function, the application may retrieve an error message describing the reason for the failure from **pmem_errormsg**(). This function returns a pointer to a static buffer containing the last error message logged for the current thread. If *errno* was set, the error message may include a description of the corresponding error code as returned by **strerror**(3). The error message buffer is thread−local; errors encountered in one thread do not affect its value in other threads. The buffer is never cleared by any library function; its content is significant only when the return value of the immediately preceding call to a **libpmem** function indicated an error, or if *errno* was set. The application must not modify or free the error message string, but it may be modified by subsequent calls to other library functions.

Two versions of **libpmem** are typically available on a development system. The normal version, accessed when a program is linked using the **−lpmem** option, is optimized for performance. That version skips checks that impact performance and never logs any trace information or performs any run−time assertions.

A second version of **libpmem**, accessed when a program uses the libraries under **/usr/lib/pmdk_debug**, contains run−time assertions and trace points. The typical way to access the debug version is to set the environment variable **LD_LIBRARY_PATH** to **/usr/lib/pmdk_debug** or **/usr/lib64/pmdk_debug**, as appropriate. Debugging output is controlled using the following environment variables. These variables have no effect on the non−debug version of the library.

- **PMEM_LOG_LEVEL**

The value of **PMEM_LOG_LEVEL** enables trace points in the debug version of the library, as follows:

- **0 −** This is the default level when **PMEM_LOG_LEVEL** is not set. No log messages are emitted at this level.
- **1 −** Additional details on any errors detected are logged, in addition to returning the *errno*−based errors as usual. The same information may be retrieved using **pmem_errormsg**().
- **2 −** A trace of basic operations is logged.
- **3 −** Enables a very verbose amount of function call tracing in the library.
- **4 −** Enables voluminous and fairly obscure tracing information that is likely only useful to the **libpmem** developers.

Unless **PMEM_LOG_FILE** is set, debugging output is written to *stderr*.

- **PMEM_LOG_FILE**

Specifies the name of a file where all logging information should be written. If the last character in the name is "−", the *PID* of the current process will be appended to the file name when the log file is created.

If **PMEM_LOG_FILE** is not set, output is written to *stderr*.

EXAMPLE

The following example uses **libpmem** to flush changes made to raw, memory−mapped persistent memory.

WARNING: There is nothing transactional about the **pmem_persist**(3) or **pmem_msync**(3) calls in this example. Interrupting the program may result in a partial write to pmem. Use a transactional library such as **libpmemobj**(7) to avoid torn updates.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <libpmem.h>

/* using 4k of pmem for this example */
#define PMEM_LEN 4096

#define PATH "/pmem-fs/myfile"

int
main(int argc, char *argv[])
{
    char *pmemaddr;
    size_t mapped_len;
    int is_pmem;

    /* create a pmem file and memory map it */

    if ((pmemaddr = pmem_map_file(PATH, PMEM_LEN, PMEM_FILE_CREATE,
            0666, &mapped_len, &is_pmem)) == NULL) {
        perror("pmem_map_file");
        exit(1);
    }

    /* store a string to the persistent memory */
    strcpy(pmemaddr, "hello, persistent memory");

    /* flush above strcpy to persistence */
    if (is_pmem)
        pmem_persist(pmemaddr, mapped_len);
    else
        pmem_msync(pmemaddr, mapped_len);

    /*
     * Delete the mappings. The region is also
     * automatically unmapped when the process is
     * terminated.
     */
    pmem_unmap(pmemaddr, mapped_len);
}
```

See <http://pmem.io/pmdk/libpmem> for more examples using the **libpmem** API.

ACKNOWLEDGEMENTS

libpmem builds on the persistent memory programming model recommended by the SNIA NVM Programming Technical Work Group: <http://snia.org/nvmp>

SEE ALSO

dlclose(3), **pmem_flush**(3), **pmem_is_pmem**(3), **pmem_memmove_persist**(3), **pmem_msync**(3), **pmem_persist**(3), **strerror**(3), **libpmemblk**(7), **libpmemlog**(7), **libpmemobj**(7) and **<http://pmem.io>**