NAME

pmem_is_pmem(), **pmem_map_file**(), **pmem_unmap**() − check persistency, create and delete mappings

SYNOPSIS

```
#include <libpmem.h>

int pmem_is_pmem(const void *addr, size_t len);
void *pmem_map_file(const char *path, size_t len, int flags,
    mode_t mode, size_t *mapped_lenp, int *is_pmemp);
int pmem_unmap(void *addr, size_t len);
```

DESCRIPTION

Most pmem−aware applications will take advantage of higher level libraries that alleviate the need for the application to call into **libpmem** directly. Application developers that wish to access raw memory mapped persistence directly (via **mmap**(2)) and that wish to take on the responsibility for flushing stores to persistence will find the functions described in this section to be the most commonly used.

The **pmem_is_pmem**() function detects if the entire range [*addr*, *addr*+*len*) consists of persistent memory. The implementation of **pmem_is_pmem**() requires a non−trivial amount of work to determine if the given range is entirely persistent memory. For this reason, it is better to call **pmem_is_pmem**() once when a range of memory is first encountered, save the result, and use the saved result to determine whether **pmem_persist**(3) or **msync**(2) is appropriate for flushing changes to persistence. Calling **pmem_is_pmem**() each time changes are flushed to persistence will not perform well.

The **pmem_map_file**() function creates a new read/write mapping for a file. If **PMEM_FILE_CREATE** is not specified in *flags*, the entire existing file *path* is mapped, *len* must be zero, and *mode* is ignored. Otherwise, *path* is opened or created as specified by *flags* and *mode*, and *len* must be non−zero. **pmem_map_file**() maps the file using **mmap**(2), but it also takes extra steps to make large page mappings more likely.

On success, **pmem_map_file**() returns a pointer to the mapped area. If *mapped_lenp* is not NULL, the length of the mapping is stored into **mapped_lenp*. If *is_pmemp* is not NULL, a flag indicating whether the mapped file is actual pmem, or if **msync**() must be used to flush writes for the mapped range, is stored into **is_pmemp*.

The *flags* argument is 0 or the bitwise OR of one or more of the following file creation flags:

- **PMEM_FILE_CREATE** − Create the file named *path* if it does not exist. *len* must be non−zero and specifies the size of the file to be created. If the file already exists, it will be extended or truncated to *len.* The new or existing file is then fully allocated to size *len* using **posix_fallocate**(3). *mode* specifies the mode to use in case a new file is created (see **creat**(2)).

The remaining flags modify the behavior of **pmem_map_file**() when **PMEM_FILE_CREATE** is specified.

- **PMEM_FILE_EXCL** − If specified in conjunction with **PMEM_FILE_CREATE**, and *path* already exists, then **pmem_map_file**() will fail with **EEXIST**. Otherwise, has the same meaning as **O_EXCL** on **open**(2), which is generally undefined.
- **PMEM_FILE_SPARSE** − When specified in conjunction with **PMEM_FILE_CREATE**, create a sparse (holey) file using **ftruncate**(2) rather than allocating it using **posix_fallocate**(3). Otherwise ignored.
- **PMEM_FILE_TMPFILE** − Create a mapping for an unnamed temporary file. Must be specified with **PMEM_FILE_CREATE**. *len* must be non−zero, *mode* is ignored (the temporary file is always created with mode 0600), and *path* must specify an existing directory name. If the underlying file system supports **O_TMPFILE**, the unnamed temporary file is created in the filesystem containing the directory *path*; if **PMEM_FILE_EXCL** is also specified, the temporary file may not subsequently be linked into the filesystem (see **open**(2)). Otherwise, the file is created in *path* and immediately unlinked.

The *path* can point to a Device DAX. In this case only the **PMEM_FILE_CREATE** and **PMEM_FILE_SPARSE** flags are valid, but they are both ignored. For Device DAX mappings, *len* must

be equal to either 0 or the exact size of the device.

To delete mappings created with **pmem_map_file**(), use **pmem_unmap**().

The **pmem_unmap**() function deletes all the mappings for the specified address range, and causes further references to addresses within the range to generate invalid memory references. It will use the address specified by the parameter *addr*, where *addr* must be a previously mapped region. **pmem_unmap**() will delete the mappings using **munmap**(2).

RETURN VALUE

The **pmem_is_pmem**() function returns true only if the entire range [*addr*, *addr*+*len*) consists of persistent memory. A true return from **pmem_is_pmem**() means it is safe to use **pmem_persist**(3) and the related functions to make changes durable for that memory range. See also **CAVEATS**.

On success, **pmem_map_file**() returns a pointer to the memory−mapped region and sets **mapped_lenp* and **is_pmemp* if they are not NULL. On error, it returns NULL, sets *errno* appropriately, and does not modify **mapped_lenp* or **is_pmemp*.

On success, **pmem_unmap**() returns 0. On error, it returns −1 and sets *errno* appropriately.

NOTES

On Linux, **pmem_is_pmem**() returns true only if the entire range is mapped directly from Device DAX (/dev/daxX.Y) without an intervening file system. In the future, as file systems become available that support flushing with **pmem_persist**(3), **pmem_is_pmem**() will return true as appropriate.

CAVEATS

The result of **pmem_is_pmem**() query is only valid for the mappings created using **pmem_map_file**(). For other memory regions, in particular those created by a direct call to **mmap**(2), **pmem_is_pmem**() always returns false, even if the queried range is entirely persistent memory.

Not all file systems support **posix_fallocate**(3). **pmem_map_file**() will fail if **PMEM_FILE_CREATE** is specified without **PMEM_FILE_SPARSE** and the underlying file system does not support **posix_fallocate**(3).

SEE ALSO

creat(2), **ftruncate**(2), **mmap**(2), **msync**(2), **munmap**(2), **open**(2), **pmem_persist**(3), **posix_fallocate**(3), **libpmem**(7) and **<http://pmem.io>**