NAME

memfd_create, **shm_create_largepage**, **shm_open**, **shm_rename, shm_unlink -** shared memory object operations

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <sys/types.h>
#include <sys/mman.h>
#include <fcntl.h>

int
memfd_create(*const char *name, unsigned int flags*);

int
shm_create_largepage(*const char *path, int flags, int psind, int alloc_policy, mode_t mode*);

int
shm_open(*const char *path, int flags, mode_t mode*);

int
shm_rename(*const char *path_from, const char *path_to, int flags*);

int
shm_unlink(*const char *path*);

DESCRIPTION

The **shm_open**() function opens (or optionally creates) a POSIX shared memory object named *path*. The *flags* argument contains a subset of the flags used by open(2). An access mode of either O_RDONLY or O_RDWR must be included in *flags*. The optional flags O_CREAT, O_EXCL, and O_TRUNC may also be specified.

If O_CREAT is specified, then a new shared memory object named *path* will be created if it does not exist. In this case, the shared memory object is created with mode *mode* subject to the process' umask value. If both the O_CREAT and O_EXCL flags are specified and a shared memory object named *path* already exists, then **shm_open**() will fail with EEXIST.

Newly created objects start off with a size of zero. If an existing shared memory object is opened with O_RDWR and the O_TRUNC flag is specified, then the shared memory object will be truncated to a

size of zero. The size of the object can be adjusted via ftruncate(2) and queried via fstat(2).

The new descriptor is set to close during execve(2) system calls; see close(2) and fcntl(2).

The constant SHM_ANON may be used for the *path* argument to **shm_open**(). In this case, an anonymous, unnamed shared memory object is created. Since the object has no name, it cannot be removed via a subsequent call to **shm_unlink**(), or moved with a call to **shm_rename**(). Instead, the shared memory object will be garbage collected when the last reference to the shared memory object is removed. The shared memory object may be shared with other processes by sharing the file descriptor via fork(2) or sendmsg(2). Attempting to open an anonymous shared memory object with O_RDONLY will fail with EINVAL. All other flags are ignored.

The **shm_create_largepage**() function behaves similarly to **shm_open**(), except that the O_CREAT flag is implicitly specified, and the returned "largepage" object is always backed by aligned, physically contiguous chunks of memory. This ensures that the object can be mapped using so-called "superpages", which can improve application performance in some workloads by reducing the number of translation lookaside buffer (TLB) entries required to access a mapping of the object, and by reducing the number of page faults performed when accessing a mapping. This happens automatically for all largepage objects.

An existing largepage object can be opened using the **shm_open**() function. Largepage shared memory objects behave slightly differently from non-largepage objects:

- Memory for a largepage object is allocated when the object is extended using the ftruncate(2) system call, whereas memory for regular shared memory objects is allocated lazily and may be paged out to a swap device when not in use.

- The size of a mapping of a largepage object must be a multiple of the underlying large page size. Most attributes of such a mapping can only be modified at the granularity of the large page size. For example, when using munmap(2) to unmap a portion of a largepage object mapping, or when using mprotect(2) to adjust protections of a mapping of a largepage object, the starting address must be large page size-aligned, and the length of the operation must be a multiple of the large page size. If not, the corresponding system call will fail and set *errno* to EINVAL.

The *psind* argument to **shm_create_largepage**() specifies the size of large pages used to back the object. This argument is an index into the page sizes array returned by getpagesizes(3). In particular, all large pages backing a largepage object must be of the same size. For example, on a system with large page sizes of 2MB and 1GB, a 2GB largepage object will consist of either 1024 2MB pages, or 2 1GB pages, depending on the value specified for the *psind* argument. The *alloc_policy* parameter specifies what

happens when an attempt to use ftruncate(2) to allocate memory for the object fails. The following values are accepted:

SHM_LARGEPAGE_ALLOC_DEFAULT
: If the (non-blocking) memory allocation fails because there is insufficient free contiguous memory, the kernel will attempt to defragment physical memory and try another allocation. The subsequent allocation may or may not succeed. If this subsequent allocation also fails, ftruncate(2) will fail and set *errno* to ENOMEM.

SHM_LARGEPAGE_ALLOC_NOWAIT
: If the memory allocation fails, ftruncate(2) will fail and set *errno* to ENOMEM.

SHM_LARGEPAGE_ALLOC_HARD
: The kernel will attempt defragmentation until the allocation succeeds, or an unblocked signal is delivered to the thread. However, it is possible for physical memory to be fragmented such that the allocation will never succeed.

The FIOSSHMLPGCNF and FIOGSHMLPGCNF ioctl(2) commands can be used with a largepage shared memory object to get and set largepage object parameters. Both commands operate on the following structure:

```
struct shm_largepage_conf {
        int psind;
        int alloc_policy;
};
```

The FIOGSHMLPGCNF command populates this structure with the current values of these parameters, while the FIOSSHMLPGCNF command modifies the largepage object. Currently only the *alloc_policy* parameter may be modified. Internally, **shm_create_largepage**() works by creating a regular shared memory object using **shm_open**(), and then converting it into a largepage object using the FIOSSHMLPGCNF ioctl command.

The **shm_rename**() system call atomically removes a shared memory object named *path_from* and relinks it at *path_to*. If another object is already linked at *path_to*, that object will be unlinked, unless one of the following flags are provided:

SHM_RENAME_EXCHANGE
: Atomically exchange the shms at *path_from* and *path_to*.

SHM_RENAME_NOREPLACE

Return an error if an shm exists at *path_to*, rather than unlinking it.

The **shm_unlink**() system call removes a shared memory object named *path*.

The **memfd_create**() function creates an anonymous shared memory object, identical to that created by **shm_open**() when SHM_ANON is specified. Newly created objects start off with a size of zero. The size of the new object must be adjusted via ftruncate(2).

The *name* argument must not be NULL, but it may be an empty string. The length of the *name* argument may not exceed NAME_MAX minus six characters for the prefix "memfd:", which will be prepended. The *name* argument is intended solely for debugging purposes and will never be used by the kernel to identify a memfd. Names are therefore not required to be unique.

The following *flags* may be specified to **memfd_create**():

MFD_CLOEXEC Set FD_CLOEXEC on the resulting file descriptor.

MFD_ALLOW_SEALING Allow adding seals to the resulting file descriptor using the F_ADD_SEALS fcntl(2) command.

MFD_HUGETLB This flag is currently unsupported.

RETURN VALUES

If successful, **memfd_create**() and **shm_open**() both return a non-negative integer, and **shm_rename**() and **shm_unlink**() return zero. All functions return -1 on failure, and set *errno* to indicate the error.

COMPATIBILITY

The **shm_create_largepage**() and **shm_rename**() functions are FreeBSD extensions, as is support for the SHM_ANON value in **shm_open**().

The *path*, *path_from*, and *path_to* arguments do not necessarily represent a pathname (although they do in most other implementations). Two processes opening the same *path* are guaranteed to access the same shared memory object if and only if *path* begins with a slash (‘/’) character.

Only the O_RDONLY, O_RDWR, O_CREAT, O_EXCL, and O_TRUNC flags may be used in portable programs.

POSIX specifications state that the result of using open(2), read(2), or write(2) on a shared memory object, or on the descriptor returned by **shm_open**(), is undefined. However, the FreeBSD kernel implementation explicitly includes support for read(2) and write(2).

FreeBSD also supports zero-copy transmission of data from shared memory objects with sendfile(2).

Neither shared memory objects nor their contents persist across reboots.

Writes do not extend shared memory objects, so ftruncate(2) must be called before any data can be written. See *EXAMPLES*.

EXAMPLES

This example fails without the call to ftruncate(2):

```
uint8_t buffer[getpagesize()];
ssize_t len;
int fd;

fd = shm_open(SHM_ANON, O_RDWR | O_CREAT, 0600);
if (fd < 0)
	err(EX_OSERR, "%s: shm_open", __func__);
if (ftruncate(fd, getpagesize()) < 0)
	err(EX_IOERR, "%s: ftruncate", __func__);
len = pwrite(fd, buffer, getpagesize(), 0);
if (len < 0)
	err(EX_IOERR, "%s: pwrite", __func__);
if (len != getpagesize())
	errx(EX_IOERR, "%s: pwrite length mismatch", __func__);
```

ERRORS

memfd_create() fails with these error codes for these conditions:

[EBADF] The *name* argument was NULL.

[EINVAL] The *name* argument was too long.

An invalid or unsupported flag was included in *flags*.

[EMFILE] The process has already reached its limit for open file descriptors.

[ENFILE] The system file table is full.

[ENOSYS] In *memfd_create*, MFD_HUGETLB was specified in *flags*, and this system does not support forced hugetlb mappings.

shm_open() fails with these error codes for these conditions:

[EINVAL] A flag other than O_RDONLY, O_RDWR, O_CREAT, O_EXCL, or O_TRUNC was included in *flags*.

[EMFILE] The process has already reached its limit for open file descriptors.

[ENFILE] The system file table is full.

[EINVAL] O_RDONLY was specified while creating an anonymous shared memory object via SHM_ANON.

[EFAULT] The *path* argument points outside the process' allocated address space.

[ENAMETOOLONG]
The entire pathname exceeds 1023 characters.

[EINVAL] The *path* does not begin with a slash ('/') character.

[ENOENT] O_CREAT is not specified and the named shared memory object does not exist.

[EEXIST] O_CREAT and O_EXCL are specified and the named shared memory object does exist.

[EACCES] The required permissions (for reading or reading and writing) are denied.

[ECAPMODE] The process is running in capability mode (see capsicum(4)) and attempted to create a named shared memory object.

shm_create_largepage() can fail for the reasons listed above. It also fails with these error codes for the following conditions:

[ENOTTY] The kernel does not support large pages on the current platform.

The following errors are defined for **shm_rename**():

[EFAULT] The *path_from* or *path_to* argument points outside the process' allocated address space.

[ENAMETOOLONG]

The entire pathname exceeds 1023 characters.

[ENOENT] The shared memory object at *path_from* does not exist.

[EACCES] The required permissions are denied.

[EEXIST] An shm exists at *path_to*, and the SHM_RENAME_NOREPLACE flag was provided.

shm_unlink() fails with these error codes for these conditions:

[EFAULT] The *path* argument points outside the process' allocated address space.

[ENAMETOOLONG]
The entire pathname exceeds 1023 characters.

[ENOENT] The named shared memory object does not exist.

[EACCES] The required permissions are denied. **shm_unlink**() requires write permission to the shared memory object.

SEE ALSO

posixshmcontrol(1), close(2), fstat(2), ftruncate(2), ioctl(2), mmap(2), munmap(2), sendfile(2)

STANDARDS

The **memfd_create**() function is expected to be compatible with the Linux system call of the same name.

The **shm_open**() and **shm_unlink**() functions are believed to conform to IEEE Std 1003.1b-1993 ("POSIX.1b").

HISTORY

The **memfd_create**() function appeared in FreeBSD 13.0.

The **shm_open**() and **shm_unlink**() functions first appeared in FreeBSD 4.3. The functions were reimplemented as system calls using shared memory objects directly rather than files in FreeBSD 8.0.

shm_rename() first appeared in FreeBSD 13.0 as a FreeBSD extension.

AUTHORS

Garrett A. Wollman <*wollman@FreeBSD.org*> (C library support and this manual page)

Matthew Dillon <*dillon@FreeBSD.org*> (MAP_NOSYNC)

Matthew Bryan <*matthew.bryan@isilon.com*> (shm_rename implementation)