Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'shm_open.3p' command

$ man shm_open.3p

SHM_OPEN(3P) POSIX Programmer's Manual SHM_OPEN(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

shm_open ? open a shared memory object (REALTIME)

SYNOPSIS

#include <sys/mman.h>

int shm_open(const char *name, int oflag, mode_t mode);

DESCRIPTION

The shm_open() function shall establish a connection between a shared memory object and a file descriptor. It shall create an open file de? scription that refers to the shared memory object and a file descriptor that refers to that open file description. The file descriptor shall be allocated as described in Section 2.14, File Descriptor Allocation, and can be used by other functions to refer to that shared memory object. The name argument points to a string naming a shared memory object. It is unspecified whether the name appears in the file system and is visi? ble to other functions that take pathnames as arguments. The name argu? ment conforms to the construction rules for a pathname, except that the interpretation of <slash> characters other than the leading <slash>

character in name is implementation-defined, and that the length limits for the name argument are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If name begins with the <slash> character, then processes calling shm_open() with the same value of name refer to the same shared memory object, as long as that name has not been removed. If name does not begin with the <slash> character, the effect is implementation-defined.

If successful, shm_open() shall return a file descriptor for the shared memory object. The open file description is new, and therefore the file descriptor does not share it with any other processes. It is un? specified whether the file offset is set. The FD_CLOEXEC file descrip? tor flag associated with the new file descriptor is set.

The file status flags and file access modes of the open file descrip? tion are according to the value of oflag. The oflag argument is the bitwise-inclusive OR of the following flags defined in the <fcntl.h> header. Applications specify exactly one of the first two values (ac? cess modes) below in the value of oflag:

O_RDONLY Open for read access only.

O_RDWR Open for read or write access.

Any combination of the remaining flags may be specified in the value of oflag:

O_CREAT If the shared memory object exists, this flag has no ef? fect, except as noted under O_EXCL below. Otherwise, the shared memory object is created. The user ID of the shared memory object shall be set to the effective user ID of the process. The group ID of the shared memory object shall be set to the effective group ID of the process; however, if the name argument is visible in the file system, the group ID may be set to the group ID of the containing directory. The permission bits of the shared memory object shall be set to the value of the mode argument except those set in the file mode creation mask of the process. When bits in mode other than the file permission bits are set, the ef?

fect is unspecified. The mode argument does not affect whether the shared memory object is opened for reading, for writing, or for both. The shared memory object has a size of zero.

O_EXCL If O_EXCL and O_CREAT are set, shm_open() fails if the shared memory object exists. The check for the existence of the shared memory object and the creation of the object if it does not exist is atomic with respect to other processes executing shm_open() naming the same shared memory object with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined.

O_TRUNC If the shared memory object exists, and it is successfully opened O_RDWR, the object shall be truncated to zero length and the mode and owner shall be unchanged by this function call. The result of using O_TRUNC with O_RDONLY is unde? fined.

When a shared memory object is created, the state of the shared memory object, including all data associated with the shared memory object, persists until the shared memory object is unlinked and all other ref? erences are gone. It is unspecified whether the name and shared memory object state remain valid after a system reboot.

RETURN VALUE

Upon successful completion, the shm_open() function shall return a non-negative integer representing the file descriptor. Otherwise, it shall return -1 and set errno to indicate the error.

ERRORS

The shm_open() function shall fail if:

EACCES The shared memory object exists and the permissions specified by oflag are denied, or the shared memory object does not exist and permission to create the shared memory object is denied, or O_TRUNC is specified and write permission is denied.

EEXIST O_CREAT and O_EXCL are set and the named shared memory object already exists.

EINTR The shm_open() operation was interrupted by a signal.

EINVAL The shm_open() operation is not supported for the given name.

EMFILE All file descriptors available to the process are currently open.

ENFILE Too many shared memory objects are currently open in the system.

ENOENT O_CREAT is not set and the named shared memory object does not exist.

ENOSPC There is insufficient space for the creation of the new shared memory object.

The shm_open() function may fail if:

ENAMETOOLONG

The length of the name argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems.

The following sections are informative.

EXAMPLES

Creating and Mapping a Shared Memory Object

The following code segment demonstrates the use of shm_open() to create a shared memory object which is then sized using ftruncate() before be? ing mapped into the process address space using mmap():

```
#include <unistd.h>
#include <sys/mman.h>
...
#define MAX_LEN 10000
struct region {        /* Defines "structure" of shared memory */
    int len;
    char buf[MAX_LEN];
};
struct region *rptr;
int fd;
```

```
/* Create shared memory object and set its size */
fd = shm_open("/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
if (fd == -1)
    /* Handle error */;
if (ftruncate(fd, sizeof(struct region)) == -1)
    /* Handle error */;
/* Map shared memory object */
rptr = mmap(NULL, sizeof(struct region),
       PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
if (rptr == MAP_FAILED)
    /* Handle error */;
/* Now we can refer to mapped region using fields of rptr;
   for example, rptr->len */
...
```

APPLICATION USAGE

None.

RATIONALE

When the Memory Mapped Files option is supported, the normal open() call is used to obtain a descriptor to a file to be mapped according to existing practice with mmap(). When the Shared Memory Objects option is supported, the shm_open() function shall obtain a descriptor to the shared memory object to be mapped.

There is ample precedent for having a file descriptor represent several types of objects. In the POSIX.1?1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory. Many implemen? tations simply have an operations vector, which is indexed by the file descriptor type and does very different operations. Note that in some cases the file descriptor passed to generic operations on file descrip? tors is returned by open() or creat() and in some cases returned by al? ternate functions, such as pipe(). The latter technique is used by shm_open().

Note that such shared memory objects can actually be implemented as mapped files. In both cases, the size can be set after the open using

ftruncate(). The shm_open() function itself does not create a shared object of a specified size because this would duplicate an extant func? tion that set the size of an object referenced by a file descriptor.

On implementations where memory objects are implemented using the ex? isting file system, the shm_open() function may be implemented using a macro that invokes open(), and the shm_unlink() function may be imple? mented using a macro that invokes unlink().

For implementations without a permanent file system, the definition of the name of the memory objects is allowed not to survive a system re? boot. Note that this allows systems with a permanent file system to im? plement memory objects as data structures internal to the implementa? tion as well.

On implementations that choose to implement memory objects using memory directly, a shm_open() followed by an ftruncate() and close() can be used to preallocate a shared memory area and to set the size of that preallocation. This may be necessary for systems without virtual memory hardware support in order to ensure that the memory is contiguous.

The set of valid open flags to shm_open() was restricted to O_RDONLY, O_RDWR, O_CREAT, and O_TRUNC because these could be easily implemented on most memory mapping systems. This volume of POSIX.1?2017 is silent on the results if the implementation cannot supply the requested file access because of implementation-defined reasons, including hardware ones.

The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the implementation cannot complete a request.

[EACCES] indicates for implementation-defined reasons, probably hard? ware-related, that the implementation cannot comply with a requested mode because it conflicts with another requested mode. An example might be that an application desires to open a memory object two times, map? ping different areas with different access modes. If the implementation cannot map a single area into a process space in two places, which would be required if different access modes were required for the two areas, then the implementation may inform the application at the time

of the second open.

[ENOTSUP] indicates for implementation-defined reasons, probably hard? ware-related, that the implementation cannot comply with a requested mode at all. An example would be that the hardware of the implementa? tion cannot support write-only shared memory areas.

On all implementations, it may be desirable to restrict the location of the memory objects to specific file systems for performance (such as a RAM disk) or implementation-defined reasons (shared memory supported directly only on certain file systems). The shm_open() function may be used to enforce these restrictions. There are a number of methods available to the application to determine an appropriate name of the file or the location of an appropriate directory. One way is from the environment via getenv(). Another would be from a configuration file.

This volume of POSIX.1?2017 specifies that memory objects have initial contents of zero when created. This is consistent with current behavior for both files and newly allocated memory. For those implementations that use physical memory, it would be possible that such implementa? tions could simply use available memory and give it to the process uninitialized. This, however, is not consistent with standard behavior for the uninitialized data area, the stack, and of course, files. Fi? nally, it is highly desirable to set the allocated memory to zero for security reasons. Thus, initializing memory objects to zero is re? quired.

FUTURE DIRECTIONS

A future version might require the shm_open() and shm_unlink() func? tions to have semantics similar to normal file system operations.

SEE ALSO

Section 2.14, File Descriptor Allocation, close(), dup(), exec, fc? ntl(), mmap(), shmat(), shmctl(), shmdt(), shm_unlink(), umask()

The Base Definitions volume of POSIX.1?2017, <fcntl.h>, <sys_mman.h>

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Por?

table Operating System Interface (POSIX), The Open Group Base Specifi? cations Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker? nel.org/doc/man-pages/reporting_bugs.html .