NAME

smcreat, smopen, smclose, smget, smput − shared memory operations

SYNOPSIS

```
#include <sys/shmem.h>

smcreat (path, access, size);
char *path;
short access;
long size;

sm_des = smopen (path, mode);
smdes_t sm_des;
char *path;
short mode;

smclose (sm_des);
smdes_t sm_des;

vaddr = smget (sm_des, mode, offset, size, time_lmt);
caddt_t vaddr;
smdes_t sm_des;
short mode;
long offset;
long size;
short time_lmt;

smput (vaddr)
caddt_t vaddr;
```

DESCRIPTION

Shared memory is a form of memory which can be attached to a process's address space, read or written, and then released, with the contents preserved for later or simultaneous attachment by another process. Shared memory can be used as multiple access user memory (MAUS), or as a means of passing large pieces of data from one process to another with only one process accessing the data at a time. Exactly how it is used is determined by the user processes. Each piece of shared memory is referenced orginally via the UNIX file system and can also be accessed as a file, via the normal *open, read, write, lseek,* and *close* system calls.

smcreat Shared memory is created dynamically with the *smcreat* system call. *Smcreat* is analogous to *creat* except that a size in bytes must be specified at creation time. Shared memory retains a fixed size during whatever time it exists as a part of the file system. If *size* is not a multiple of **BSIZE** (See **/usr/include/sys/param.h**) bytes it is rounded up. *Path* is a pointer to a normal UNIX pathname. *Access* specifies in the normal way (See *chmod*(2)) who may *open* or *smopen* a specific piece of shared memory.

If the specified piece of shared memory already exists and no process has it attached and this user has the proper permissions, the old shared memory will be recreated to the new size. All newly created shared memory is initialized to all zeros.

smopen Once a piece of shared memory exists, it can be opened for attachment to a process's address space via the *smopen* system call. *Path* is again a UNIX pathname specifying which piece of shared memory. *Mode* is the normal file system type mode, 0 for read, 1 for write, 2 for read and write permissions. *Write,* actually means *read-write* on most systems, since write permissions on memory imply read permissions. *sm_des* (shared memory descriptor) is returned upon a successful *smopen,* and is used when attaching shared memory via the *smget* system call.

smclose *Smclose* releases a specified *sm_des*. Any sections attached at the time of the *smclose* will remain attached, but once detached, will not be accessable again without a new *smopen*.

smget *Smget* performs the actual attachment of shared memory to the user process's address space. *Sm_des* is a shared memory descriptor previously returned from *sm_creat* or *sm_open*. *Mode* specifies how the user wishes to access the memory once it is attached. There are three modes as defined in **<sys/shmem.h>**:

SMREAD
Attach the memory read only.

SMWRITE
Attach the memory for reading and writing. Note that there is no equivalent of 'write only'.

SMREADLOCK
Attach the memory for reading only, but only when no one else has it attached for writing. If someone else has it attached for writing, wait. Do not allow new requests for writing to succeed. This means that eventually the request will succeed. **SMREADLOCK** guarantees that the data being read is stable.

SMWRITELOCK
Attach the memory for reading and writing, but only when there is no one else who has this section attached for writing. If other people have any portion of this section attached for writing, wait for the time period specified by *time_lmt* while they drain away. Do not allow any new people to attach the requested section for writing so that eventually the *smget* will succeed. Using the **SMWRITELOCK** feature to access shared memory removes the requirement for any outside locking procedures such as semaphores. The user is guaranteed that when *smget* with a mode of **SMWRITELOCK** succeeds, that this is the only process writing that memory at this time. Until an *smput* is done on this section of memory, it will remain so locked.

The following state table shows the interactions between the current state of a piece of shared memory and a new request to have it attached with a particular mode via *smget*. **T** means request will be granted immediately. **F** means requester will have to wait.

Requested State	Current State				
—	r	w	rl	wl	unattached
r	T	T	T	T	T
w	T	T	F	F	T
rl	T	F	T	F	T
wl	T	F	F	F	T

Offset is an unsigned offset from the beginning of a piece of shared memory to which a user wishes to attach. It should be some multiple of **BSIZE** bytes. If it is not, the *smget* will fail. *Size* is the unsigned size of the section of shared memory that the process wishes to attach. It also should be a multiple of **BSIZE** bytes. It may be rounded up by as much as **BSIZE** −1 bytes so that the entire section of shared memory the user requested access to is available. *Smget* will fail if the section of shared memory requested is not within the limits of the piece of shared

memory as it was created by *smcreat*. *Time_lmt* is the amount of time the user is willing to wait until a particular section of shared memory is free, when trying to do an *smget* with a mode of **SMWRITELOCK**. If *time_lmt* is 0, the *smget* will fail immediately if anyone else has any portion of the requested shared memory attached. If *time_lmt* is less than 0, the user is willing to wait indefinately for the section of shared memory to become free. If *time_lmt* is positive, then the user is willing to wait this many seconds for the section of shared memory to become available. If *smget* fails because the *time_lmt* was exceeded, the error **EBUSY** will be returned in *errno* and *vaddr* will be set to **NULL**. Whenever *smget* succeeds, *vaddr* will be some multiple of **BSIZE** bytes and is the pointer to the section of attached shared memory.

If *sm_des* is **SMDESNONE** (As defined in <sys/**shmem.h**>), the *smget* behaves like a memory allocator. An unnamed section of memory, **size** big, is attached to the user process. *Mode* is only meaningful if it is **SMWRITELOCK**. This mode will prevent a child process from inheriting the attached section of memory. In other cases the section of memory is readable and writable. Memory attached in this fashion is guaranteed to be zeroed. *Offset* is ignored. *Time_lmt* behaves in the normal fashion.

smput *Smput* releases an attached section of shared memory. *Vaddr* must match the value returned by *smget*.

A copy of **/usr/include/sys/shmem.h** is included here for reference.

```
/*      "mode" definitions to be used with "smget".                    */

#define                 SMREAD          0
#define                 SMWRITE         1
#define                 SMREADLOCK      2
#define                 SMWRITELOCK     3

/*      Shared memory descriptor to use when attaching                 */
/*      unnamed memory.                                                */

#define                 SMDESNONE   (-1)

typedef     short       smdes_t ;

#ifdef      KERNEL

/*      Structure maintaining the state of each page of                */
/*      shared memory.                                                 */

struct SM_pgstate
 {
        daddr_t sm_block ;          /* Block for this page
                                     * into the file attached
                                     * as shared memory.
                                     * Are stored in ascending
                                     * order for any file.
                                     */
        struct buf *sm_bufpt ;      /* Ptr to buffer header
                                     * for this page.
                                     */
        char sm_refcnt ;            /* Count of total users
                                     * attached to this page.
                                     */
        char sm_rlrefcnt ;          /* Count of users attached to
                                     * this page as read locked.
                                     */
```

```
                char sm_wrefcnt ;           /* Count of users attached
                                             * to this page for writing.
                                             */
                char sm_wlwant ;            /* Number of users wanting page
                                             * write locked.
                                             */
                char sm_rlwant ;            /* Number of users wanting page
                                             * read locked.
                                             */
                char sm_wwant ;             /* Count of users wanting page
                                             * for writing.
                                             */
                short sm_flags ;
                struct SM_pgstate *sm_next ;  /* Pointer to next page's
                                               * control structure.
                                               */
        } ;

        /*      Definitions for "sm_flags".                                  */

        #define         IS_OCCUPIED         1
        #define         IS_WRITELOCKED      2
        #define         IS_READLOCKED       4
        #define         WL_REQUEST          10
        #define         RL_REQUEST          20
        #define         W_REQUEST           40

        struct SM_pgptrs
         {
                struct SM_pgstate *s_current ;
                struct SM_pgstate *s_previous ;
         } ;
        #endif
```

FILES

/dev/shmem/*

SHARED MEMORY RULES

1) Shared memory descriptors are inherited across forks and executes.

2) Sections of attached shared memory are inherited across forks if they were not opened **SMWRITELOCK.** Writelocked sections are retained by the parent, but closed to the child, keeping them writelocked.

3) Attached sections of shared memory are not inherited across *exec's.*

4) If a *break* system call tries to expand memory into an attached section of shared memory, it will fail.

5) If some process tries to *open* a shared memory file while another process is waiting for an *smget* with **SMWRITELOCK** to succeed, the *open* will fail.

SEE ALSO

break(2), close(2), creat(2), open(2)

DIAGNOSTICS

From assembly code, the carry bit is set in the case of errors and *errno* set with an indication of the specific error. From C a −1 is returned from *smcreat, smclose, smopen,* and *smput,* and a **NULL** from *smget.*

IMPLEMENTATION CONSIDERATIONS

It is envisioned that a shared memory file will be a special file type. It will be implemented only under version 7 or later file systems. Each section of shared memory will require two inodes, a visible inode referenced in the UNIX file system, and an invisible inode, used only by

the shared memory routines to access the data when it is on the disk. This implementation will require that *check* understand this new file type and not remove the invisible inode during a check.

It should be noted that the implementation of *smget* and *smput* interact very nicely with the *MSG* implementation proposed by Dale DeJager. When passing large *no_copy* messages, memory must first be allocated and the final receiver must return it to the operating system. *Smget* and *smput* nicely serve the purpose of the routines *memget* and *memfree*.

When new shared memory is created initially, a control block will be allocated at the same time. In the control block will be an *SM_control* structure for each page of this section of shared memory. The structure contains two counts, the count of the total number of users attached to this page, and the number of users attached to this page for writing. It also contains five flags, **IS_WRITELOCKED,** meaning that the page is currently writelocked, **IS_READLOCKED,** meaning that the page is currently readlocked, **WL_REQUEST,** meaning that someone is requesting writelock permissions for this page, **RL_REQUEST,** meaning that someone is requesting readlock permission for this page, and **W_REQUEST,** meaning that someone is requesting write permission for this page. When a process requests a page for readlocked access, each page will be locked starting from the beginning, if that page has a 0 reference count for writers. If the refenerce count is something other than 0, the process will set the **RL_REQUEST** flag and sleep on the page until it is awakened and finds the writing reference count 0. When a process requests a page for writelocked access, each page will be locked starting from the beginning, if that page has a 0 reference count for writers and has the **IS_READLOCKED** and **IS_WRITELOCKED** flags off. If either condition isn't met, the process will sleep on the page until it awakens to find both conditions satisfied. If someone is requesting read access, they always succeed. If someone is requesting write access, they will succeed if the page isn't **IS_WRITELOCKED** or **IS_READLOCKED.** If they can't attach immediately they will set the **W_REQUEST** and sleep on the page until both conditions are satisfied before succeeding. In all cases, if the *time_lmt* expires during the wait for the pages to become available, the request will fail.