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 'shmdt.2' command

$ man shmdt.2

SHMOP(2) Linux Programmer's Manual SHMOP(2)

NAME

shmat, shmdt - System V shared memory operations

SYNOPSIS

#include <sys/types.h>

#include <sys/shm.h>

void *shmat(int shmid, const void *shmaddr, int shmflg);

int shmdt(const void *shmaddr);

DESCRIPTION

shmat()

shmat() attaches the System V shared memory segment identified by shmid to the address space of the calling process. The attaching address is specified by shmaddr with one of the following criteria:

? If shmaddr is NULL, the system chooses a suitable (unused) page-aligned address to attach the segment.

? If shmaddr isn't NULL and SHM_RND is specified in shmflg, the attach occurs at the address equal to shmaddr rounded down to the nearest multiple of SHMLBA.

? Otherwise, shmaddr must be a page-aligned address at which the attach occurs.

In addition to SHM_RND, the following flags may be specified in the shmflg bit-mask argument:

SHM_EXEC (Linux-specific; since Linux 2.6.9)

Allow the contents of the segment to be executed. The caller must have execute permission on the segment.

SHM_RDONLY

Attach the segment for read-only access. The process must have read permission for the segment. If this flag is not specified, the segment is attached for read and write access, and the process must have read and write permission for the segment. There is no notion of a write-only shared memory segment.

SHM_REMAP (Linux-specific)

This flag specifies that the mapping of the segment should re? place any existing mapping in the range starting at shmaddr and continuing for the size of the segment. (Normally, an EINVAL error would result if a mapping already exists in this address range.) In this case, shmaddr must not be NULL.

The brk(2) value of the calling process is not altered by the attach.

The segment will automatically be detached at process exit. The same segment may be attached as a read and as a read-write one, and more than once, in the process's address space.

A successful shmat() call updates the members of the shmid_ds structure (see shmctl(2)) associated with the shared memory segment as follows:

? shm_atime is set to the current time.

? shm_lpid is set to the process-ID of the calling process.

? shm_nattch is incremented by one.

shmdt()

shmdt() detaches the shared memory segment located at the address spec? ified by shmaddr from the address space of the calling process. The to-be-detached segment must be currently attached with shmaddr equal to the value returned by the attaching shmat() call.

On a successful shmdt() call, the system updates the members of the shmid_ds structure associated with the shared memory segment as fol? lows:

? shm_dtime is set to the current time.

? shm_lpid is set to the process-ID of the calling process.

? shm_nattch is decremented by one. If it becomes 0 and the segment is marked for deletion, the segment is deleted.

RETURN VALUE

On success, shmat() returns the address of the attached shared memory segment; on error, (void *) -1 is returned, and errno is set to indi? cate the cause of the error.

On success, shmdt() returns 0; on error -1 is returned, and errno is set to indicate the cause of the error.

ERRORS

When shmat() fails, errno is set to one of the following:

EACCES The calling process does not have the required permissions for the requested attach type, and does not have the CAP_IPC_OWNER capability in the user namespace that governs its IPC namespace.

EIDRM shmid points to a removed identifier.

EINVAL Invalid shmid value, unaligned (i.e., not page-aligned and SHM_RND was not specified) or invalid shmaddr value, or can't attach segment at shmaddr, or SHM_REMAP was specified and shmaddr was NULL.

ENOMEM Could not allocate memory for the descriptor or for the page ta? bles.

When shmdt() fails, errno is set as follows:

EINVAL There is no shared memory segment attached at shmaddr; or, shmaddr is not aligned on a page boundary.

CONFORMING TO

POSIX.1-2001, POSIX.1-2008, SVr4.

In SVID 3 (or perhaps earlier), the type of the shmaddr argument was changed from char * into const void *, and the returned type of shmat() from char * into void *.

NOTES

After a fork(2), the child inherits the attached shared memory seg? ments.

After an execve(2), all attached shared memory segments are detached from the process.

Upon _exit(2), all attached shared memory segments are detached from the process.

Using shmat() with shmaddr equal to NULL is the preferred, portable way of attaching a shared memory segment. Be aware that the shared memory segment attached in this way may be attached at different addresses in different processes. Therefore, any pointers maintained within the shared memory must be made relative (typically to the starting address of the segment), rather than absolute.

On Linux, it is possible to attach a shared memory segment even if it is already marked to be deleted. However, POSIX.1 does not specify this behavior and many other implementations do not support it.

The following system parameter affects shmat():

SHMLBA Segment low boundary address multiple. When explicitly specify? ing an attach address in a call to shmat(), the caller should ensure that the address is a multiple of this value. This is necessary on some architectures, in order either to ensure good CPU cache performance or to ensure that different attaches of the same segment have consistent views within the CPU cache. SHMLBA is normally some multiple of the system page size. (On many Linux architectures, SHMLBA is the same as the system page size.)

The implementation places no intrinsic per-process limit on the number of shared memory segments (SHMSEG).

EXAMPLES

The two programs shown below exchange a string using a shared memory segment. Further details about the programs are given below. First, we show a shell session demonstrating their use.

In one terminal window, we run the "reader" program, which creates a System V shared memory segment and a System V semaphore set. The pro? gram prints out the IDs of the created objects, and then waits for the semaphore to change value.

```
$ ./svshm_string_read
shmid = 1114194; semid = 15
```

In another terminal window, we run the "writer" program. The "writer" program takes three command-line arguments: the IDs of the shared mem? ory segment and semaphore set created by the "reader", and a string. It attaches the existing shared memory segment, copies the string to the shared memory, and modifies the semaphore value.

```
$ ./svshm_string_write 1114194 15 'Hello, world'
```

Returning to the terminal where the "reader" is running, we see that the program has ceased waiting on the semaphore and has printed the string that was copied into the shared memory segment by the writer:

```
Hello, world
```

Program source: svshm_string.h

The following header file is included by the "reader" and "writer" pro? grams.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define errExit(msg)    do { perror(msg); exit(EXIT_FAILURE); \
                        } while (0)
union semun {                   /* Used in calls to semctl() */
    int                 val;
    struct semid_ds *   buf;
    unsigned short *    array;
#if defined(__linux__)
    struct seminfo *    __buf;
#endif
};
#define MEM_SIZE 4096
```

Program source: svshm_string_read.c

The "reader" program creates a shared memory segment and a semaphore set containing one semaphore. It then attaches the shared memory ob? ject into its address space and initializes the semaphore value to 1. Finally, the program waits for the semaphore value to become 0, and af? terwards prints the string that has been copied into the shared memory segment by the "writer".

```
/* svshm_string_read.c
   Licensed under GNU General Public License v2 or later.
*/
#include "svshm_string.h"
int
main(int argc, char *argv[])
{
    int semid, shmid;
    union semun arg, dummy;
    struct sembuf sop;
    char *addr;
    /* Create shared memory and semaphore set containing one
       semaphore */
    shmid = shmget(IPC_PRIVATE, MEM_SIZE, IPC_CREAT | 0600);
    if (shmid == -1)
        errExit("shmget");
    semid = semget(IPC_PRIVATE, 1, IPC_CREAT | 0600);
    if (shmid == -1)
        errExit("shmget");
    /* Attach shared memory into our address space */
    addr = shmat(shmid, NULL, SHM_RDONLY);
    if (addr == (void *) -1)
        errExit("shmat");
    /* Initialize semaphore 0 in set with value 1 */
    arg.val = 1;
    if (semctl(semid, 0, SETVAL, arg) == -1)
        errExit("semctl");
    printf("shmid = %d; semid = %d\n", shmid, semid);
```

```
    /* Wait for semaphore value to become 0 */
    sop.sem_num = 0;
    sop.sem_op = 0;
    sop.sem_flg = 0;
    if (semop(semid, &sop, 1) == -1)
        errExit("semop");
    /* Print the string from shared memory */
    printf("%s\n", addr);
    /* Remove shared memory and semaphore set */
    if (shmctl(shmid, IPC_RMID, NULL) == -1)
        errExit("shmctl");
    if (semctl(semid, 0, IPC_RMID, dummy) == -1)
        errExit("semctl");
    exit(EXIT_SUCCESS);
}
```

Program source: svshm_string_write.c

The writer program takes three command-line arguments: the IDs of the shared memory segment and semaphore set that have already been created by the "reader", and a string. It attaches the shared memory segment into its address space, and then decrements the semaphore value to 0 in order to inform the "reader" that it can now examine the contents of the shared memory.

```
/* svshm_string_write.c
   Licensed under GNU General Public License v2 or later.
*/
#include "svshm_string.h"
int
main(int argc, char *argv[])
{
    int semid, shmid;
    struct sembuf sop;
    char *addr;
    size_t len;
```

```
    if (argc != 4) {
        fprintf(stderr, "Usage: %s shmid semid string\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    len = strlen(argv[3]) + 1;  /* +1 to include trailing '\0' */
    if (len > MEM_SIZE) {
        fprintf(stderr, "String is too big!\n");
        exit(EXIT_FAILURE);
    }
    /* Get object IDs from command-line */
    shmid = atoi(argv[1]);
    semid = atoi(argv[2]);
    /* Attach shared memory into our address space and copy string
       (including trailing null byte) into memory. */
    addr = shmat(shmid, NULL, 0);
    if (addr == (void *) -1)
        errExit("shmat");
    memcpy(addr, argv[3], len);
    /* Decrement semaphore to 0 */
    sop.sem_num = 0;
    sop.sem_op = -1;
    sop.sem_flg = 0;
    if (semop(semid, &sop, 1) == -1)
        errExit("semop");
    exit(EXIT_SUCCESS);
}
```

SEE ALSO

brk(2), mmap(2), shmctl(2), shmget(2), capabilities(7), shm_over?
view(7), sysvipc(7)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.