

NAME

shmget - get shared memory segment identifier

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int shmget(key_t key, size_t size, int shmflg);
```

DESCRIPTION

The shmget() function returns the shared memory identifier associated with `key`.

A shared memory identifier and associated data structure and shared memory segment of at least `size` bytes (see intro(3)) are created for `key` if one of the following are true:

- + o The `key` argument is equal to `IPC_PRIVATE`.
- + o The `key` argument does not already have a shared memory identifier associated with it, and `(shmflg & IPC_CREAT)` is true.

Upon creation, the data structure associated with the new shared memory identifier is initialized as follows:

- + o The values of `shm_perm.cuid`, `shm_perm.uid`, `shm_perm.cgid`, and `shm_perm.gid` are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- + o The access permission bits of `shm_perm.mode` are set equal to the access permission bits of `shmflg`. `shm_segsz` is set equal to the value of `size`.
- + o The values of `shm_lpid`, `shm_nattch`, `shm_atime`, and `shm_dtime` are set equal to 0.
- + o The `shm_ctime` is set equal to the current time.

Shared memory segments must be explicitly removed after the last reference to them has been removed.

RETURN VALUES

Upon successful completion, a non-negative integer representing a shared memory identifier is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS

The `shmget()` function will fail if:

EACCES

A shared memory identifier exists for `_k_e_y` but operation permission (see `intro(3)`) as specified by the low-order 9 bits of `_s_h_m_f_l_g` would not be granted.

EEXIST

A shared memory identifier exists for `_k_e_y` but both `(_s_h_m_f_l_g&IPC_CREATE)` and `(_s_h_m_f_l_g&IPC_EXCL)` are true.

EINVAL

The `_s_i_z_e` argument is less than the system-imposed minimum or greater than the system-imposed maximum.

EINVAL

A shared memory identifier exists for `_k_e_y` but the size of the segment associated with it is less than `_s_i_z_e` and `_s_i_z_e` is not equal to 0.

ENOENT

A shared memory identifier does not exist for `_k_e_y` and `(_s_h_m_f_l_g&IPC_CREATE)` is false.

ENOMEM

A shared memory identifier and associated shared memory segment are to be created but the amount of available memory is not sufficient to fill the request.

ENOSPC

A shared memory identifier is to be created but the system-imposed limit on the maximum number of allowed shared memory identifiers system-wide would be exceeded.

System Calls

`shmop(2)`

NAME

`shmop`, `shmat`, `shmdt` - shared memory operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/shm.h>
```

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

Default

```
int shmdt(char *shmaddr);
```

Standard-conforming

int shmdt(const void *shmaddr);

DESCRIPTION

The shmat() function attaches the shared memory segment associated with the shared memory identifier specified by `shm_id` to the data segment of the calling process.

The permission required for a shared memory control operation is given as `{type}`, where `type` is the type of permission needed. The types of permission are interpreted as follows:

00400 READ by user
00200 WRITE by user
00040 READ by group
00020 WRITE by group
00004 READ by others
00002 WRITE by others

See the `Shared Memory Operations` section of `intro(2)` for more information.

When `SHM_SHARE_MMU` is true, virtual memory resources in addition to shared memory itself are shared among processes that use the same shared memory.

The shared memory segment is attached to the data segment of the calling process at the address specified based on one of the following criteria:

- + o If `shm_addr` is equal to `(void *) 0`, the segment is attached to the first available address as selected by the system.
- + o If `shm_addr` is equal to `(void *) 0` and `SHM_SHARE_MMU` is true, then the segment is attached to the first available suitably aligned address. When `SHM_SHARE_MMU` is set, however, the permission given by `shmget()` determines whether the segment is attached for reading or reading and writing.
- + o If `shm_addr` is not equal to `(void *) 0` and `SHM_RND` is true, the segment is attached to the address given by $(shm_addr - shm_addr \bmod SHMLBA)$.
- + o If `shm_addr` is not equal to `(void *) 0` and `SHM_RND` is false, the segment is attached to the address given by `shm_addr`.
- + o The segment is attached for reading if

(`_s_h_m_f_l_g&SHM_RDONLY`) is true {READ}, otherwise it is attached for reading and writing {READ/WRITE}.

The `shmdt()` function detaches from the calling process's data segment the shared memory segment located at the address specified by `_s_h_m_a_d_d_r`. If the application is standard-conforming (see standards(5)), the `_s_h_m_a_d_d_r` argument is of type `const void *`. Otherwise it is of type `char *`.

Shared memory segments must be explicitly removed after the last reference to them has been removed.

RETURN VALUES

Upon successful completion, `shmat()` returns the data segment start address of the attached shared memory segment; `shmdt()` returns 0. Otherwise, -1 is returned, the shared memory segment is not attached, and `errno` is set to indicate the error.

ERRORS

The `shmat()` function will fail if:

EACCESS

Operation permission is denied to the calling process (see `intro(2)`).

EINVAL

The `_s_h_m_i_d` argument is not a valid shared memory identifier.

EINVAL

The `_s_h_m_a_d_d_r` argument is not equal to 0, and the value of (`_s_h_m_a_d_d_r` modulus `SHMLBA`) is an illegal address.

EINVAL

The `_s_h_m_a_d_d_r` argument is not equal to 0, is an illegal address, and (`_s_h_m_f_l_g&SHM_RND`) is false.

EINVAL

The `_s_h_m_a_d_d_r` argument is not equal to 0, is not properly aligned, and (`_s_h_m_f_l_g&SHM_SHARE_MMU`) is true.

EINVAL

`SHM_SHARE_MMU` is not supported in certain architectures.

EMFILE

The number of shared memory segments attached to the calling process would exceed the system-imposed limit.

ENOMEM

The available data space is not large enough to accommodate the shared memory segment.

The `shmdt()` function will fail if:

EINVAL

The `_s_h_m_a_d_` `d_r` argument is not the data segment start address of a shared memory segment.

System Calls shmctl(2)

NAME

shmctl - shared memory control operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

DESCRIPTION

The `shmctl()` function provides a variety of shared memory control operations as specified by `_c_m_d`. The permission required for a shared memory control operation is given as `{_t_o_k_e_n}`, where `_t_o_k_e_n` is the type of permission needed. The types of permission are interpreted as follows:

```
00400  READ by user
00200  WRITE by user
00040  READ by group
00020  WRITE by group
00004  READ by others
00002  WRITE by others
```

See the `_S_h_a_r_e_d_M_e_m_o_r_y_O_p_e_r_a_t_i_o_n_s` section of `_P_e_r_m_i_s_s_i_o_n_s` intro(2) for more information.

The following operations require the specified tokens:

IPC_STAT

Place the current value of each member of the data structure associated with `_s_h_m_` `i_d` into the structure pointed to by `_b_u_f`. The contents of this structure are defined in intro(2). {READ}

IPC_SET

Set the value of the following members of the data structure associated with `_s_h_m_` `i_d` to the corresponding value found in the structure pointed to by `_b_u_f`:

```
shm_perm.uid
shm_perm.gid
shm_perm.mode /* access permission bits only */
```

This command can be executed only by a process that has an effective user ID equal to that of super-user, or to the value of `shm_perm.cuid` or `shm_perm.uid` in the data structure associated with `_s_h_m_ i_d`.

IPC_RMID

Remove the shared memory identifier specified by `_s_h_m_ i_d` from the system and destroy the shared memory segment and data structure associated with it. This command can be executed only by a process that has an effective user ID equal to that of super-user, or to the value of `shm_perm.cuid` or `shm_perm.uid` in the data structure associated with `_s_h_m_ i_d`.

SHM_LOCK

Lock the shared memory segment specified by `_s_h_m_ i_d` in memory. This command can be executed only by a process that has an effective user ID equal to super-user.

SHM_UNLOCK

Unlock the shared memory segment specified by `_s_h_m_ i_d`. This command can be executed only by a process that has an effective user ID equal to super-user.

Shared memory segments must be explicitly removed after the last reference to them has been removed.

RETURN VALUES

Upon successful completion, 0 is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS

The `shmctl()` function will fail if:

EACCES

The `_c_m_d` argument is equal to `IPC_STAT` and `{READ}` operation permission is denied to the calling process.

EFAULT

The `_b_u_f` argument points to an illegal address.

EINVAL

The `_s_h_m_ i_d` argument is not a valid shared memory identifier; or the `_c_m_d` argument is not a valid command or is `IPC_SET` and `shm_perm.uid` or `shm_perm.gid` is not valid.

ENOMEM

The `_c_m_d` argument is equal to `SHM_LOCK` and there is not enough memory.

EOVERFLOW

The `_c_m_d` argument is `IPC_STAT` and `_u_ i_d` or `_g_ i_d` is too large to be stored in the structure pointed to by `_b_u_f`.

EPERM The `_c_m_d` argument is equal to `IPC_RMID` or `IPC_SET` and

the effective user ID of the calling process is not super-user and it is not equal to the value of shm_perm.cuid or shm_perm.uid in the data structure associated with shm_id.

EPERM The cmd argument is equal to SHM_LOCK or SHM_UNLOCK and the effective user ID of the calling process is not equal to that of super-user.

Standard C Library Functions

memory(3C)

NAME

memory, memccpy, memchr, memcmp, memcpy, memmove, memset -
memory operations

SYNOPSIS

```
#include <string.h>
```

```
void *memccpy(void *s1, const void *s2, int c, size_t n);
```

```
void *memchr(const void *s, int c, size_t n);
```

```
int memcmp(const void *s1, const void *s2, size_t n);
```

```
void *memcpy(void *s1, const void *s2, size_t n);
```

```
void *memmove(void *s1, const void *s2, size_t n);
```

```
void *memset(void *s, int c, size_t n);
```

ISO C++

```
#include <string.h>
```

```
const void *memchr(const void *s, int c, size_t n);
```

```
#include <cstring>
```

```
void *std::memchr(void *s, int c, size_t n);
```

DESCRIPTION

These functions operate as efficiently as possible on memory areas (arrays of bytes bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

The `memccpy()` function copies bytes from memory area `_s_2` into `_s_1`, stopping after the first occurrence of `_c` (converted to an unsigned char) has been copied, or after `_n` bytes have been copied, whichever comes first. It returns a pointer to the byte after the copy of `_c` in `_s_1`, or a null pointer if `_c` was not found in the first `_n` bytes of `_s_2`.

The `memchr()` function returns a pointer to the first occurrence of `_c` (converted to an unsigned char) in the first `_n` bytes (each interpreted as an unsigned char) of memory area `_s`, or a null pointer if `_c` does not occur.

The `memcmp()` function compares its arguments, looking at the first `_n` bytes (each interpreted as an unsigned char), and returns an integer less than, equal to, or greater than 0, according as `_s_1` is lexicographically less than, equal to, or greater than `_s_2` when taken to be unsigned characters.

The `memcpy()` function copies `_n` bytes from memory area `_s_2` to `_s_1`. It returns `_s_1`.

The `memmove()` function copies `_n` bytes from memory areas `_s_2` to `_s_1`. Copying between objects that overlap will take place correctly. It returns `_s_1`.

The `memset()` function sets the first `_n` bytes in memory area `_s` to the value of `_c` (converted to an unsigned char). It returns `_s`.

Standard C Library Functions

string(3C)

NAME

`string`, `strcasecmp`, `strncasemp`, `strcat`, `strncat`, `strlcat`,
`strchr`, `strrchr`, `strcmp`, `strncmp`, `strcpy`, `strncpy`, `strncpy`,
`strlcpy`,
`strcspn`, `strspn`, `strdup`, `strlen`, `strpbrk`, `strstr`, `strtok`,
`strtok_r`, - string operations

SYNOPSIS

```
#include <strings.h>
```

```
int strcasemp(const char *s1, const char *s2);
```

```
int strncasecmp(const char *s1, const char *s2, size_t n);

#include <string.h>

char *strcat(char *s1, const char *s2);

char *strncat(char *s1, const char *s2, size_t n);

size_t strlcat(char *dst, const char *src, size_t dstsize);

char *strchr(const char *s, int c);

char *strrchr(const char *s, int c);

int strcmp(const char *s1, const char *s2);

int strncmp(const char *s1, const char *s2, size_t n);

char *strcpy(char *s1, const char *s2);

char *strncpy(char *s1, const char *s2, size_t n);

size_t strlen(const char *s);

size_t strlcpy(char *dst, const char *src, size_t dstsize);

size_t strcspn(const char *s1, const char *s2);

size_t strspn(const char *s1, const char *s2);

char *strdup(const char *s1);

size_t strlen(const char *s);

char *strpbrk(const char *s1, const char *s2);

char *strstr(const char *s1, const char *s2);
```

```
char *strtok(char *s1, const char *s2);
```

```
char *strtok_r(char *s1, const char *s2, char **lasts);
```

ISO C++

```
#include <string.h>
```

```
const char *strchr(const char *s, int c);
```

```
const char *strpbrk(const char *s1, const char *s2);
```

```
const char *strrchr(const char *s, int c);
```

```
const char *strstr(const char *s1, const char *s2);
```

```
#include <cstring>
```

```
char *std::strchr(char *s, int c);
```

```
char *std::strpbrk(char *s1, const char *s2);
```

```
char *std::strrchr(char *s, int c);
```

```
char *std::strstr(char *s1, const char *s2);
```

DESCRIPTION

The arguments `_s`, `_s_1`, and `_s_2` point to strings (arrays of characters terminated by a null character). The `strcat()`, `strncat()`, `strlcat()`, `strcpy()`, `strncpy()`, `strlcpy()`, `strtok()`, and `strtok_r()` functions all alter their first argument. These functions do not check for overflow of the array pointed to by the first argument.

`strcasecmp()`, `strncasecmp()`

The `strcasecmp()` and `strncasecmp()` functions are case-insensitive versions of `strcmp()` and `strncmp()` respectively, described below. They assume the ASCII character set and ignore differences in case when comparing lower and

upper case characters.

strcat(), strncat(), strlcat()

The strcat() function appends a copy of string `_s_2`, including the terminating null character, to the end of string `_s_1`. The strncat() function appends at most `_n` characters. Each returns a pointer to the null-terminated result. The initial character of `_s_2` overrides the null character at the end of `_s_1`.

The strlcat() function appends at most $(_d_s_t_s_i_z_e - \text{strlen}(_d_s_t) - 1)$ characters of `_s_r_c` to `_d_s_t` (`_d_s_t_s_i_z_e` being the size of the string buffer `_d_s_t`). The initial character of `_s_r_c` overrides the null character at the end of `_d_s_t`. The function returns the sum of the lengths of the two strings $\text{strlen}(_d_s_t) + \text{strlen}(_s_r_c)$.

Buffer overflow can be checked as follows:

```
if (strlcat(dst, src, dstsize) >= dstsize)
    return -1;
```

strchr(), strrchr()

The strchr() function returns a pointer to the first occurrence of `_c` (converted to a char) in string `_s`, or a null pointer if `_c` does not occur in the string. The strrchr() function returns a pointer to the last occurrence of `_c`. The null character terminating a string is considered to be part of the string.

strcmp(), strncmp()

The strcmp() function compares two strings byte-by-byte, according to the ordering of your machine's character set. The function returns an integer greater than, equal to, or less than 0, if the string pointed to by `_s_1` is greater than, equal to, or less than the string pointed to by `_s_2` respectively. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes that differ in the strings being compared. The strncmp() function makes the same comparison but looks at a maximum of `_n` bytes. Bytes following a null byte are not compared.

strcpy(), strncpy(), strlcpy()

The strcpy() function copies string `_s_2` to `_s_1`, including the terminating null character, stopping after the null character has been copied. The strncpy() function copies exactly `_n` bytes, truncating `_s_2` or adding null characters to `_s_1` if necessary. The result will not be null-terminated if the length of `_s_2` is `_n` or more. Each function returns `_s_1`.

The strlcpy() function copies at most $_d_s_t_s_i_z_e - 1$ characters (`_d_s_t_s_i_z_e` being the size of the string buffer `_d_s_t`) from `_s_r_c` to `_d_s_t`, truncating `_s_r_c` if necessary. The result is always null-terminated. The function returns $\text{strlen}(_s_r_c)$. Buffer overflow can be checked as follows:

```
if (strncpy(dst, src, dstsize) >= dstsize)
    return -1;
```

strcspn(), strspn()

The strcspn() function returns the length of the initial segment of string `_s_1` that consists entirely of characters not from string `_s_2`. The strspn() function returns the length of the initial segment of string `_s_1` that consists entirely of characters from string `_s_2`.

strdup()

The strdup() function returns a pointer to a new string that is a duplicate of the string pointed to by `_s_1`. The space for the new string is obtained using malloc(3C). If the new

string cannot be created, a null pointer is returned.

strlen()

The strlen() function returns the number of bytes in `_s`, not including the terminating null character.

strpbrk()

The strpbrk() function returns a pointer to the first occurrence in string `_s_1` of any character from string `_s_2`, or a null pointer if no character from `_s_2` exists in `_s_1`.

strstr()

The strstr() function locates the first occurrence of the string `_s_2` (excluding the terminating null character) in string `_s_1` and returns a pointer to the located string, or a null pointer if the string is not found. If `_s_2` points to a string with zero length (that is, the string ""), the function returns `_s_1`.

strtok()

The strtok() function can be used to break the string pointed to by `_s_1` into a sequence of tokens, each of which is delimited by one or more characters from the string pointed to by `_s_2`. The strtok() function considers the string `_s_1` to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string `_s_2`. The first call (with pointer `_s_1` specified) returns a pointer to the first character of the first token, and will have written a null character into `_s_1` immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument being a null pointer) will work through the string `_s_1` immediately following that token. In this way subsequent calls will work through the string `_s_1` until no tokens remain. The separator string `_s_2` may be different from call to call. When no token remains in `_s_1`, a null pointer is returned.

strtok_r()

The strtok_r() function has the same functionality as

strtok() except that a pointer to a string placeholder `_l_a_s_t_s` must be supplied by the caller. The `_l_a_s_t_s` pointer is to keep track of the next substring in which to search for the next token.

NOTES

The `strtok_r()` function is as proposed in the POSIX.4a Draft #6 document, and is subject to change to be compliant to the standard when it is accepted.

When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

All of these functions assume the default locale ```C```. For some locales, `strxfrm()` should be applied to the strings before they are passed to the functions.

The `strcasemp()`, `strcat()`, `strchr()`, `strcmp()`, `strcpy()`, `strcspn()`, `strdup()`, `strlen()`, `strncasemp()`, `strncat()`, `strncmp()`, `strncpy()`, `strpbrk()`, `strrchr()`, `strspn()`, and `strstr()` functions are MT-Safe in multithreaded applications.

The `strtok()` function is Unsafe in multithreaded applications. The `strtok_r()` function should be used instead.