NAME

semget − get a System V semaphore set identifier

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#include <sys/sem.h>

int semget(key_t *key***, int** *nsems***, int** *semflg***);**

DESCRIPTION

The **semget**() system call returns the System V semaphore set identifier associated with the argument *key*. It may be used either to obtain the identifier of a previously created semaphore set (when *semflg* is zero and *key* does not have the value **IPC_PRIVATE**), or to create a new set.

A new set of *nsems* semaphores is created if *key* has the value **IPC_PRIVATE** or if no existing semaphore set is associated with *key* and **IPC_CREAT** is specified in *semflg*.

If *semflg* specifies both **IPC_CREAT** and **IPC_EXCL** and a semaphore set already exists for *key*, then **semget**() fails with *errno* set to **EEXIST**. (This is analogous to the effect of the combination **O_CREAT | O_EXCL** for **open**(2).)

Upon creation, the least significant 9 bits of the argument *semflg* define the permissions (for owner, group, and others) for the semaphore set. These bits have the same format, and the same meaning, as the *mode* argument of **open**(2) (though the execute permissions are not meaningful for semaphores, and write permissions mean permission to alter semaphore values).

When creating a new semaphore set, **semget**() initializes the set's associated data structure, *semid_ds* (see **semctl**(2)), as follows:

- *sem_perm.cuid* and *sem_perm.uid* are set to the effective user ID of the calling process.
- *sem_perm.cgid* and *sem_perm.gid* are set to the effective group ID of the calling process.
- The least significant 9 bits of *sem_perm.mode* are set to the least significant 9 bits of *semflg*.
- *sem_nsems* is set to the value of *nsems*.
- *sem_otime* is set to 0.
- *sem_ctime* is set to the current time.

The argument *nsems* can be 0 (a don't care) when a semaphore set is not being created. Otherwise, *nsems* must be greater than 0 and less than or equal to the maximum number of semaphores per semaphore set (**SEMMSL**).

If the semaphore set already exists, the permissions are verified.

RETURN VALUE

On success, **semget**() returns the semaphore set identifier (a nonnegative integer). On failure, −1 is returned, and *errno* is set to indicate the error.

ERRORS

EACCES
: A semaphore set exists for *key*, but the calling process does not have permission to access the set, and does not have the **CAP_IPC_OWNER** capability in the user namespace that governs its IPC namespace.

EEXIST
: **IPC_CREAT** and **IPC_EXCL** were specified in *semflg*, but a semaphore set already exists for *key*.

EINVAL
: *nsems* is less than 0 or greater than the limit on the number of semaphores per semaphore set (**SEMMSL**).

EINVAL
A semaphore set corresponding to *key* already exists, but *nsems* is larger than the number of semaphores in that set.

ENOENT
No semaphore set exists for *key* and *semflg* did not specify **IPC_CREAT**.

ENOMEM
A semaphore set has to be created but the system does not have enough memory for the new data structure.

ENOSPC
A semaphore set has to be created but the system limit for the maximum number of semaphore sets (**SEMMNI**), or the system wide maximum number of semaphores (**SEMMNS**), would be exceeded.

STANDARDS

SVr4, POSIX.1-2001.

NOTES

IPC_PRIVATE isn't a flag field but a *key_t* type. If this special value is used for *key*, the system call ignores all but the least significant 9 bits of *semflg* and creates a new semaphore set (on success).

Semaphore initialization

The values of the semaphores in a newly created set are indeterminate. (POSIX.1-2001 and POSIX.1-2008 are explicit on this point, although POSIX.1-2008 notes that a future version of the standard may require an implementation to initialize the semaphores to 0.) Although Linux, like many other implementations, initializes the semaphore values to 0, a portable application cannot rely on this: it should explicitly initialize the semaphores to the desired values.

Initialization can be done using **semctl**(2) **SETVAL** or **SETALL** operation. Where multiple peers do not know who will be the first to initialize the set, checking for a nonzero *sem_otime* in the associated data structure retrieved by a **semctl**(2) **IPC_STAT** operation can be used to avoid races.

Semaphore limits

The following limits on semaphore set resources affect the **semget**() call:

SEMMNI
System-wide limit on the number of semaphore sets. Before Linux 3.19, the default value for this limit was 128. Since Linux 3.19, the default value is 32,000. On Linux, this limit can be read and modified via the fourth field of */proc/sys/kernel/sem*.

SEMMSL
Maximum number of semaphores per semaphore ID. Before Linux 3.19, the default value for this limit was 250. Since Linux 3.19, the default value is 32,000. On Linux, this limit can be read and modified via the first field of */proc/sys/kernel/sem*.

SEMMNS
System-wide limit on the number of semaphores: policy dependent (on Linux, this limit can be read and modified via the second field of */proc/sys/kernel/sem*). Note that the number of semaphores system-wide is also limited by the product of **SEMMSL** and **SEMMNI**.

BUGS

The name choice **IPC_PRIVATE** was perhaps unfortunate, **IPC_NEW** would more clearly show its function.

EXAMPLES

The program shown below uses **semget**() to create a new semaphore set or retrieve the ID of an existing set. It generates the *key* for **semget**() using **ftok**(3). The first two command-line arguments are used as the *pathname* and *proj_id* arguments for **ftok**(3). The third command-line argument is an integer that specifies the *nsems* argument for **semget**(). Command-line options can be used to specify the**IPC_CREA T** (*−c*) and **IPC_EXCL** (*−x*) flags for the call to **semget**(). The usage of this program is demonstrated below.

We first create two files that will be used to generate keys using **ftok**(3), create two semaphore sets using those files, and then list the sets using **ipcs**(1):

```
$ touch mykey mykey2
$ ./t_semget -c mykey p 1
ID = 9
$ ./t_semget -c mykey2 p 2
ID = 10
$ ipcs -s

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x7004136d 9          mtk        600        1
0x70041368 10         mtk        600        2
```

Next, we demonstrate that when **semctl**(2) is given the same *key* (as generated by the same arguments to **ftok**(3)), it returns the ID of the already existing semaphore set:

```
$ ./t_semget -c mykey p 1
ID = 9
```

Finally, we demonstrate the kind of collision that can occur when **ftok**(3) is given different *pathname* arguments that have the same inode number:

```
$ ln mykey link
$ ls -i1 link mykey
2233197 link
2233197 mykey
$ ./t_semget link p 1       # Generates same key as 'mykey'
ID = 9
```

Program source

```
/* t_semget.c

   Licensed under GNU General Public License v2 or later.
*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>

static void
usage(const char *pname)
{
    fprintf(stderr, "Usage: %s [-cx] pathname proj-id num-sems\n",
            pname);
    fprintf(stderr, "    -c           Use IPC_CREAT flag\n");
    fprintf(stderr, "    -x           Use IPC_EXCL flag\n");
    exit(EXIT_FAILURE);
}

int
main(int argc, char *argv[])
{
    int    semid, nsems, flags, opt;
    key_t  key;
```

```
        flags = 0;
        while ((opt = getopt(argc, argv, "cx")) != -1) {
            switch (opt) {
            case 'c': flags |= IPC_CREAT;   break;
            case 'x': flags |= IPC_EXCL;    break;
            default:  usage(argv[0]);
            }
        }

        if (argc != optind + 3)
            usage(argv[0]);

        key = ftok(argv[optind], argv[optind + 1][0]);
        if (key == -1) {
            perror("ftok");
            exit(EXIT_FAILURE);
        }

        nsems = atoi(argv[optind + 2]);

        semid = semget(key, nsems, flags | 0600);
        if (semid == -1) {
            perror("semget");
            exit(EXIT_FAILURE);
        }

        printf("ID = %d\n", semid);

        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

semctl(2), **semop**(2), **ftok**(3), **capabilities**(7), **sem_overview**(7), **sysvipc**(7)