## NAME

semop, semtimedop − semaphore operations

## SYNOPSIS

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

int semop(int semid, struct sembuf *sops, unsigned nsops);

int semtimedop(int semid, struct sembuf *sops, unsigned nsops,
        struct timespec *timeout);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

**semtimedop**(): _GNU_SOURCE

## DESCRIPTION

Each semaphore in a semaphore set has the following associated values:

```
unsigned short  semval;   /* semaphore value */
unsigned short  semzcnt;  /* # waiting for zero */
unsigned short  semncnt;  /* # waiting for increase */
pid_t           sempid;   /* process that did last op */
```

**semop**() performs operations on selected semaphores in the set indicated by *semid*. Each of the *nsops* elements in the array pointed to by *sops* specifies an operation to be performed on a single semaphore. The elements of this structure are of type *struct sembuf*, containing the following members:

```
unsigned short sem_num;  /* semaphore number */
short          sem_op;   /* semaphore operation */
short          sem_flg;  /* operation flags */
```

Flags recognized in *sem_flg* are **IPC_NOWAIT** and **SEM_UNDO**. If an operation specifies **SEM_UNDO**, it will be automatically undone when the process terminates.

The set of operations contained in *sops* is performed in *array order*, and *atomically*, that is, the operations are performed either as a complete unit, or not at all. The behavior of the system call if not all operations can be performed immediately depends on the presence of the **IPC_NOWAIT** flag in the individual *sem_flg* fields, as noted below.

Each operation is performed on the *sem_num*−th semaphore of the semaphore set, where the first semaphore of the set is numbered 0. There are three types of operation, distinguished by the value of *sem_op*.

If *sem_op* is a positive integer, the operation adds this value to the semaphore value (*semval*). Furthermore, if **SEM_UNDO** is specified for this operation, the system updates the process undo count (*semadj*) for this semaphore. This operation can always proceed — it never forces a process to wait. The calling process must have alter permission on the semaphore set.

If *sem_op* is zero, the process must have read permission on the semaphore set. This is a "wait-for-zero" operation: if *semval* is zero, the operation can immediately proceed. Otherwise, if **IPC_NOWAIT** is specified in *sem_flg*, **semop**() fails with *errno* set to **EAGAIN** (and none of the operations in *sops* is performed). Otherwise *semzcnt* (the count of processes waiting until this semaphore's value becomes zero) is incremented by one and the process sleeps until one of the following occurs:

- *semval* becomes 0, at which time the value of *semzcnt* is decremented.

- The semaphore set is removed: **semop**() fails, with *errno* set to **EIDRM**.
- The calling process catches a signal: the value of *semzcnt* is decremented and **semop**() fails, with *errno* set to **EINTR**.
- The time limit specified by *timeout* in a **semtimedop**() call expires: **semop**() fails, with *errno* set to **EAGAIN**.

If *sem_op* is less than zero, the process must have alter permission on the semaphore set. If *semval* is greater than or equal to the absolute value of *sem_op*, the operation can proceed immediately: the absolute value of *sem_op* is subtracted from *semval*, and, if **SEM_UNDO** is specified for this operation, the system updates the process undo count (*semadj*) for this semaphore. If the absolute value of *sem_op* is greater than *semval*, and **IPC_NOWAIT** is specified in *sem_flg*, **semop**() fails, with *errno* set to **EAGAIN** (and none of the operations in *sops* is performed). Otherwise *semncnt* (the counter of processes waiting for this semaphore's value to increase) is incremented by one and the process sleeps until one of the following occurs:

- *semval* becomes greater than or equal to the absolute value of *sem_op*, at which time the value of *semncnt* is decremented, the absolute value of *sem_op* is subtracted from *semval* and, if **SEM_UNDO** is specified for this operation, the system updates the process undo count (*semadj*) for this semaphore.
- The semaphore set is removed from the system: **semop**() fails, with *errno* set to **EIDRM**.
- The calling process catches a signal: the value of *semncnt* is decremented and **semop**() fails, with *errno* set to **EINTR**.
- The time limit specified by *timeout* in a **semtimedop**() call expires: the system call fails, with *errno* set to **EAGAIN**.

On successful completion, the *sempid* value for each semaphore specified in the array pointed to by *sops* is set to the process ID of the calling process. In addition, the *sem_otime* is set to the current time.

**semtimedop**() behaves identically to **semop**() except that in those cases were the calling process would sleep, the duration of that sleep is limited by the amount of elapsed time specified by the *timespec* structure whose address is passed in the *timeout* argument. If the specified time limit has been reached, **semtimedop**() fails with *errno* set to **EAGAIN** (and none of the operations in *sops* is performed). If the *timeout* argument is NULL, then **semtimedop**() behaves exactly like **semop**().

## RETURN VALUE

If successful **semop**() and **semtimedop**() return 0; otherwise they return −1 with *errno* indicating the error.

## ERRORS

On failure, *errno* is set to one of the following:

**E2BIG** The argument *nsops* is greater than **SEMOPM**, the maximum number of operations allowed per system call.

**EACCES**
The calling process does not have the permissions required to perform the specified semaphore operations, and does not have the **CAP_IPC_OWNER** capability.

**EAGAIN**
An operation could not proceed immediately and either **IPC_NOWAIT** was specified in *sem_flg* or the time limit specified in *timeout* expired.

**EFAULT**
An address specified in either the *sops* or the *timeout* argument isn't accessible.

**EFBIG**
For some operation the value of *sem_num* is less than 0 or greater than or equal to the number of semaphores in the set.

**EIDRM**
The semaphore set was removed.

**EINTR**
While blocked in this system call, the process caught a signal; see **signal**(7).

**EINVAL**
The semaphore set doesn't exist, or *semid* is less than zero, or *nsops* has a non-positive value.

**ENOMEM**
The *sem_flg* of some operation specified **SEM_UNDO** and the system does not have enough memory to allocate the undo structure.

**ERANGE**
For some operation *sem_op+semval* is greater than **SEMVMX**, the implementation dependent maximum value for *semval*.

## VERSIONS

**semtimedop**() first appeared in Linux 2.5.52, and was subsequently backported into kernel 2.4.22. Glibc support for **semtimedop**() first appeared in version 2.3.3.

## CONFORMING TO

SVr4, POSIX.1-2001.

## NOTES

The *sem_undo* structures of a process aren't inherited by the child produced by **fork**(2), but they are inherited across an **execve**(2) system call.

**semop**() is never automatically restarted after being interrupted by a signal handler, regardless of the setting of the **SA_RESTART** flag when establishing a signal handler.

*semadj* is a per-process integer which is simply the (negative) count of all semaphore operations performed specifying the **SEM_UNDO** flag. When a semaphore's value is directly set using the **SETVAL** or **SETALL** request to **semctl**(2), the corresponding *semadj* values in all processes are cleared.

The *semval*, *sempid*, *semzcnt*, and *semnct* values for a semaphore can all be retrieved using appropriate **semctl**(2) calls.

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

**SEMOPM**
Maximum number of operations allowed for one **semop**() call (32) (on Linux, this limit can be read and modified via the third field of */proc/sys/kernel/sem*).

**SEMVMX**
Maximum allowable value for *semval*: implementation dependent (32767).

The implementation has no intrinsic limits for the adjust on exit maximum value (**SEMAEM**), the system wide maximum number of undo structures (**SEMMNU**) and the per-process maximum number of undo entries system parameters.

## BUGS

When a process terminates, its set of associated *semadj* structures is used to undo the effect of all of the semaphore operations it performed with the **SEM_UNDO** flag. This raises a difficulty: if one (or more) of these semaphore adjustments would result in an attempt to decrease a semaphore's value below zero, what should an implementation do? One possible approach would be to block until all the semaphore adjustments could be performed. This is however undesirable since it could force process termination to block for arbitrarily long periods. Another possibility is that such semaphore adjustments could be ignored altogether (somewhat analogously to failing when **IPC_NOWAIT** is specified for a semaphore operation). Linux adopts a third approach: decreasing the semaphore value as far as possible (i.e., to zero) and allowing process termination to proceed immediately.

In kernels 2.6.x, x <= 10, there is a bug that in some circumstances prevents a process that is waiting for a semaphore value to become zero from being woken up when the value does actually become zero. This bug is fixed in kernel 2.6.11.

## EXAMPLE

The following code segment uses **semop**() to atomically wait for the value of semaphore 0 to become zero, and then increment the semaphore value by one.

```
struct sembuf sops[2];
int semid;

/* Code to set semid omitted */

sops[0].sem_num = 0;        /* Operate on semaphore 0 */
sops[0].sem_op = 0;         /* Wait for value to equal 0 */
sops[0].sem_flg = 0;

sops[1].sem_num = 0;        /* Operate on semaphore 0 */
sops[1].sem_op = 1;         /* Increment value by one */
sops[1].sem_flg = 0;

if (semop(semid, sops, 2) == -1) {
    perror("semop");
    exit(EXIT_FAILURE);
}
```

## SEE ALSO

**semctl**(2), **semget**(2), **sigaction**(2), **capabilities**(7), **sem_overview**(7), **svipc**(7), **time**(7)

## COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.