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 'semop.3p' command

$ man semop.3p

SEMOP(3P) POSIX Programmer's Manual SEMOP(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

semop ? XSI semaphore operations

SYNOPSIS

```
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, size_t nsops);
```

DESCRIPTION

The semop() function operates on XSI semaphores (see the Base Defini? tions volume of POSIX.1?2017, Section 4.17, Semaphore). It is unspeci? fied whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8, Realtime.

The semop() function shall perform atomically a user-defined array of semaphore operations in array order on the set of semaphores associated with the semaphore identifier specified by the argument semid.

The argument sops is a pointer to a user-defined array of semaphore op? eration structures. The implementation shall not modify elements of this array unless the application uses implementation-defined exten? sions.

The argument nsops is the number of such structures in the array.

Each structure, sembuf, includes the following members:

```
??????????????????????????????????????????????????????????
? Member Type   ? Member Name   ?       Description        ?
??????????????????????????????????????????????????????????
?unsigned short ? sem_num       ?Semaphore number.         ?
?short          ? sem_op        ?Semaphore operation.      ?
?short          ? sem_flg       ?Operation flags.          ?
??????????????????????????????????????????????????????????
```

Each semaphore operation specified by sem_op is performed on the corre?
sponding semaphore specified by semid and sem_num.

The variable sem_op specifies one of three semaphore operations:

1. If sem_op is a negative integer and the calling process has alter
   permission, one of the following shall occur:
   * If semval(see <sys/sem.h>) is greater than or equal to the ab?
     solute value of sem_op, the absolute value of sem_op is sub?
     tracted from semval. Also, if (sem_flg &SEM_UNDO) is non-zero,
     the absolute value of sem_op shall be added to the semadj value
     of the calling process for the specified semaphore.
   * If semval is less than the absolute value of sem_op and
     (sem_flg &IPC_NOWAIT) is non-zero, semop() shall return immedi?
     ately.
   * If semval is less than the absolute value of sem_op and
     (sem_flg &IPC_NOWAIT) is 0, semop() shall increment the semncnt
     associated with the specified semaphore and suspend execution
     of the calling thread until one of the following conditions oc?
     curs:
     -- The value of semval becomes greater than or equal to the
        absolute value of sem_op. When this occurs, the value of
        semncnt associated with the specified semaphore shall be
        decremented, the absolute value of sem_op shall be sub?
        tracted from semval and, if (sem_flg &SEM_UNDO) is non-
        zero, the absolute value of sem_op shall be added to the

semadj value of the calling process for the specified sema?
phore.

-- The semid for which the calling thread is awaiting action is removed from the system. When this occurs, errno shall be set to [EIDRM] and -1 shall be returned.

-- The calling thread receives a signal that is to be caught. When this occurs, the value of semncnt associated with the specified semaphore shall be decremented, and the calling thread shall resume execution in the manner prescribed in sigaction().

2. If sem_op is a positive integer and the calling process has alter permission, the value of sem_op shall be added to semval and, if (sem_flg &SEM_UNDO) is non-zero, the value of sem_op shall be sub? tracted from the semadj value of the calling process for the speci? fied semaphore.

3. If sem_op is 0 and the calling process has read permission, one of the following shall occur:

* If semval is 0, semop() shall return immediately.

* If semval is non-zero and (sem_flg &IPC_NOWAIT) is non-zero, semop() shall return immediately.

* If semval is non-zero and (sem_flg &IPC_NOWAIT) is 0, semop() shall increment the semzcnt associated with the specified sema? phore and suspend execution of the calling thread until one of the following occurs:

-- The value of semval becomes 0, at which time the value of semzcnt associated with the specified semaphore shall be decremented.

-- The semid for which the calling thread is awaiting action is removed from the system. When this occurs, errno shall be set to [EIDRM] and -1 shall be returned.

-- The calling thread receives a signal that is to be caught. When this occurs, the value of semzcnt associated with the specified semaphore shall be decremented, and the calling

thread shall resume execution in the manner prescribed in sigaction().

Upon successful completion, the value of sempid for each semaphore specified in the array pointed to by sops shall be set to the process ID of the calling process. Also, the sem_otime timestamp shall be set to the current time, as described in Section 2.7.1, IPC General De? scription.

RETURN VALUE

Upon successful completion, semop() shall return 0; otherwise, it shall return -1 and set errno to indicate the error.

ERRORS

The semop() function shall fail if:

E2BIG The value of nsops is greater than the system-imposed maximum.

EACCES Operation permission is denied to the calling process; see Sec? tion 2.7, XSI Interprocess Communication.

EAGAIN The operation would result in suspension of the calling process but (sem_flg &IPC_NOWAIT) is non-zero.

EFBIG The value of sem_num is greater than or equal to the number of semaphores in the set associated with semid.

EIDRM The semaphore identifier semid is removed from the system.

EINTR The semop() function was interrupted by a signal.

EINVAL The value of semid is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the system-imposed limit.

ENOSPC The limit on the number of individual processes requesting a SEM_UNDO would be exceeded.

ERANGE An operation would cause a semval to overflow the system-imposed limit, or an operation would cause a semadj value to overflow the system-imposed limit.

The following sections are informative.

EXAMPLES

Setting Values in Semaphores

The following example sets the values of the two semaphores associated

with the semid identifier to the values contained in the sb array.

```
#include <sys/sem.h>
...
int semid;
struct sembuf sb[2];
int nsops = 2;
int result;
/* Code to initialize semid. */
...
/* Adjust value of semaphore in the semaphore array semid. */
sb[0].sem_num = 0;
sb[0].sem_op = -1;
sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
sb[1].sem_num = 1;
sb[1].sem_op = 1;
sb[1].sem_flg = 0;
result = semop(semid, sb, nsops);
```

Creating a Semaphore Identifier

The following example gets a unique semaphore key using the ftok() function, then gets a semaphore ID associated with that key using the semget() function (the first call also tests to make sure the semaphore exists). If the semaphore does not exist, the program creates it, as shown by the second call to semget(). In creating the semaphore for the queuing process, the program attempts to create one semaphore with read/write permission for all. It also uses the IPC_EXCL flag, which forces semget() to fail if the semaphore already exists.

After creating the semaphore, the program uses calls to semctl() and semop() to initialize it to the values in the sbuf array. The number of processes that can execute concurrently without queuing is initially set to 2. The final call to semget() creates a semaphore identifier that can be used later in the program.

Processes that obtain semid without creating it check that sem_otime is non-zero, to ensure that the creating process has completed the semop()

initialization.

The final call to semop() acquires the semaphore and waits until it is free; the SEM_UNDO option releases the semaphore when the process ex? its, waiting until there are less than two processes running concur? rently.

```
#include <stdio.h>
#include <sys/sem.h>
#include <sys/stat.h>
#include <errno.h>
#include <stdlib.h>
...
key_t semkey;
int semid;
struct sembuf sbuf;
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
} arg;
struct semid_ds ds;
...
/* Get unique key for semaphore. */
if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
    perror("IPC error: ftok"); exit(1);
}
/* Get semaphore ID associated with this key. */
if ((semid = semget(semkey, 0, 0)) == -1) {
    /* Semaphore does not exist - Create. */
    if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
        S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
    {
        /* Initialize the semaphore. */
        arg.val = 0;
```

```
        sbuf.sem_num = 0;
        sbuf.sem_op = 2;  /* This is the number of runs without queuing. */
        sbuf.sem_flg = 0;
        if (semctl(semid, 0, SETVAL, arg) == -1
            || semop(semid, &sbuf, 1) == -1) {
            perror("IPC error: semop"); exit(1);
        }
    }
    else if (errno == EEXIST) {
        if ((semid = semget(semkey, 0, 0)) == -1) {
            perror("IPC error 1: semget"); exit(1);
        }
        goto check_init;
    }
    else {
        perror("IPC error 2: semget"); exit(1);
    }
}
else
{
    /* Check that semid has completed initialization. */
    /* An application can use a retry loop at this point rather than
       exiting. */
    check_init:
    arg.buf = &ds;
    if (semctl(semid, 0, IPC_STAT, arg) < 0) {
        perror("IPC error 3: semctl"); exit(1);
    }
    if (ds.sem_otime == 0) {
        perror("IPC error 4: semctl"); exit(1);
    }
}
...
```

```
    sbuf.sem_num = 0;
    sbuf.sem_op = -1;
    sbuf.sem_flg = SEM_UNDO;
    if (semop(semid, &sbuf, 1) == -1) {
        perror("IPC Error: semop"); exit(1);
    }
```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for inter? process communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7, XSI Interprocess Communication can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, XSI Interprocess Communication, Section 2.8, Realtime, exec, exit(), fork(), semctl(), semget(), sem_close(), sem_destroy(), sem_getvalue(), sem_init(), sem_open(), sem_post(), sem_trywait(), sem_unlink()

The Base Definitions volume of POSIX.1?2017, Section 4.17, Semaphore, <sys_ipc.h>, <sys_sem.h>, <sys_types.h>

COPYRIGHT

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .