



## ***Rocky Enterprise Linux 9.2 Manual Pages on command 'sem\_trywait.3'***

**C:~>man sem\_trywait.3**

SEM\_WAIT(3)                      Linux Programmer's Manual                      SEM\_WAIT(3)

### NAME

sem\_wait, sem\_timedwait, sem\_trywait - lock a semaphore

### SYNOPSIS

```
#include <semaphore.h>
```

```
int sem_wait(sem_t *sem);
```

```
int sem_trywait(sem_t *sem);
```

```
int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout);
```

Link with -pthread.

Feature Test Macro Requirements for glibc (see feature\_test\_macros(7)):

```
sem_timedwait(): _POSIX_C_SOURCE >= 200112L
```

### DESCRIPTION

sem\_wait() decrements (locks) the semaphore pointed to by sem. If the semaphore's value is greater than zero, then the decrement proceeds, and the function returns, immediately. If the semaphore currently has the value zero, then the call blocks until either it becomes possible to perform the decrement (i.e., the semaphore value rises above zero), or a signal handler interrupts the call.

sem\_trywait() is the same as sem\_wait(), except that if the decrement cannot be immediately performed, then call returns an error (errno set to EAGAIN) instead of blocking.

sem\_timedwait() is the same as sem\_wait(), except that abs\_timeout specifies a limit on the amount of time that the call should block if the decrement cannot be

immediately performed. The `abs_timeout` argument points to a structure that specifies an absolute timeout in seconds and nanoseconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC). This structure is defined as follows:

```
struct timespec {
    time_t tv_sec;    /* Seconds */
    long tv_nsec;    /* Nanoseconds [0 .. 999999999] */
};
```

If the timeout has already expired by the time of the call, and the semaphore could not be locked immediately, then `sem_timedwait()` fails with a timeout error (`errno` set to `ETIMEDOUT`).

If the operation can be performed immediately, then `sem_timedwait()` never fails with a timeout error, regardless of the value of `abs_timeout`. Furthermore, the validity of `abs_timeout` is not checked in this case.

#### RETURN VALUE

All of these functions return 0 on success; on error, the value of the semaphore is left unchanged, -1 is returned, and `errno` is set to indicate the error.

#### ERRORS

**EINTR** The call was interrupted by a signal handler; see `signal(7)`.

**EINVAL** `sem` is not a valid semaphore.

The following additional error can occur for `sem_trywait()`:

**EAGAIN** The operation could not be performed without blocking (i.e., the semaphore currently has the value zero).

The following additional errors can occur for `sem_timedwait()`:

**EINVAL** The value of `abs_timeout.tv_nsec` is less than 0, or greater than or equal to 1000 million.

#### ETIMEDOUT

The call timed out before the semaphore could be locked.

#### ATTRIBUTES

For an explanation of the terms used in this section, see `attributes(7)`.

??

?Interface            ? Attribute   ? Value   ?

??

?`sem_wait()`, `sem_trywait()`, ? Thread safety ? MT-Safe ?

?sem\_timedwait() ? ? ?

??

CONFORMING TO

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

EXAMPLE

The (somewhat trivial) program shown below operates on an unnamed semaphore. The program expects two command-line arguments. The first argument specifies a seconds value that is used to set an alarm timer to generate a SIGALRM signal. This handler performs a sem\_post(3) to increment the semaphore that is being waited on in main() using sem\_timedwait(). The second command-line argument specifies the length of the timeout, in seconds, for sem\_timedwait(). The following shows what happens on two different runs of the program:

```
$ ./a.out 2 3
About to call sem_timedwait()
sem_post() from handler
sem_timedwait() succeeded

$ ./a.out 2 1
About to call sem_timedwait()
sem_timedwait() timed out
```

Program source

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <time.h>
#include <assert.h>
#include <errno.h>
#include <signal.h>

sem_t sem;

#define handle_error(msg) \
do { perror(msg); exit(EXIT_FAILURE); } while (0)

static void
handler(int sig)
```

```

{
write(STDOUT_FILENO, "sem_post() from handler\n", 24);
if (sem_post(&sem) == -1) {
write(STDERR_FILENO, "sem_post() failed\n", 18);
_exit(EXIT_FAILURE);
}
}
int
main(int argc, char *argv[])
{
struct sigaction sa;
struct timespec ts;
int s;
if (argc != 3) {
fprintf(stderr, "Usage: %s <alarm-secs> <wait-secs>\n",
argv[0]);
exit(EXIT_FAILURE);
}
if (sem_init(&sem, 0, 0) == -1)
handle_error("sem_init");
/* Establish SIGALRM handler; set alarm timer using argv[1] */
sa.sa_handler = handler;
sigemptyset(&sa.sa_mask);
sa.sa_flags = 0;
if (sigaction(SIGALRM, &sa, NULL) == -1)
handle_error("sigaction");
alarm(atoi(argv[1]));
/* Calculate relative interval as current time plus
number of seconds given argv[2] */
if (clock_gettime(CLOCK_REALTIME, &ts) == -1)
handle_error("clock_gettime");
ts.tv_sec += atoi(argv[2]);
printf("main() about to call sem_timedwait()\n");

```

```

while ((s = sem_timedwait(&sem, &ts)) == -1 && errno == EINTR)
    continue;    /* Restart if interrupted by handler */

/* Check what happened */
if (s == -1) {
    if (errno == ETIMEDOUT)
        printf("sem_timedwait() timed out\n");
    else
        perror("sem_timedwait");
} else
    printf("sem_timedwait() succeeded\n");

exit((s == 0) ? EXIT_SUCCESS : EXIT_FAILURE);
}

```

#### SEE ALSO

clock\_gettime(2), sem\_getvalue(3), sem\_post(3), sem\_overview(7), time(7)

#### COLOPHON

This page is part of release 5.05 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/>.