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 'sem_timedwait.3p' command

$ man sem_timedwait.3p

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

sem_timedwait ? lock a semaphore

SYNOPSIS

```
#include <semaphore.h>
#include <time.h>
int sem_timedwait(sem_t *restrict sem,
    const struct timespec *restrict abstime);
```

DESCRIPTION

The sem_timedwait() function shall lock the semaphore referenced by sem as in the sem_wait() function. However, if the semaphore cannot be locked without waiting for another process or thread to unlock the sem? aphore by performing a sem_post() function, this wait shall be termi? nated when the specified timeout expires.

The timeout shall expire when the absolute time specified by abstime passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds abstime), or if the ab? solute time specified by abstime has already been passed at the time of

the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the clock on which it is based. The timespec data type is defined as a structure in the <time.h> header.

Under no circumstance shall the function fail with a timeout if the semaphore can be locked immediately. The validity of the abstime need not be checked if the semaphore can be locked immediately.

RETURN VALUE

The sem_timedwait() function shall return zero if the calling process successfully performed the semaphore lock operation on the semaphore designated by sem. If the call was unsuccessful, the state of the sem? aphore shall be unchanged, and the function shall return a value of -1 and set errno to indicate the error.

ERRORS

The sem_timedwait() function shall fail if:

EINVAL The process or thread would have blocked, and the abstime param? eter specified a nanoseconds field value less than zero or greater than or equal to 1000 million.

ETIMEDOUT

The semaphore could not be locked before the specified timeout expired.

The sem_timedwait() function may fail if:

EDEADLK

A deadlock condition was detected.

EINTR A signal interrupted this function.

EINVAL The sem argument does not refer to a valid semaphore.

The following sections are informative.

EXAMPLES

The program shown below operates on an unnamed semaphore. The program expects two command-line arguments. The first argument specifies a sec? onds 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().

```
    #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;
    static void
    handler(int sig)
    {
        int sav_errno = errno;
        static const char info_msg[] = "sem_post() from handler\n";
        write(STDOUT_FILENO, info_msg, sizeof info_msg - 1);
        if (sem_post(&sem) == -1) {
            static const char err_msg[] = "sem_post() failed\n";
            write(STDERR_FILENO, err_msg, sizeof err_msg - 1);
            _exit(EXIT_FAILURE);
        }
        errno = sav_errno;
    }
    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) {
    perror("sem_init");
    exit(EXIT_FAILURE);
}
/* 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) {
    perror("sigaction");
    exit(EXIT_FAILURE);
}
alarm(atoi(argv[1]));
/* Calculate relative interval as current time plus
   number of seconds given argv[2] */
if (clock_gettime(CLOCK_REALTIME, &ts) == -1) {
    perror("clock_gettime");
    exit(EXIT_FAILURE);
}
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);
    }
```

APPLICATION USAGE

Applications using these functions may be subject to priority inver? sion, as discussed in the Base Definitions volume of POSIX.1?2017, Sec? tion 3.291, Priority Inversion.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sem_post(), sem_trywait(), semctl(), semget(), semop(), time()

The Base Definitions volume of POSIX.1?2017, Section 3.291, Priority Inversion, <semaphore.h>, <time.h>