NAME

sem_wait, sem_trywait – acquire or wait for a semaphore

SYNOPSIS

cc [*flag* ...] *file* ... **–lposix4** [*library* ...]

#include <semaphore.h>

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

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

```
typedef struct {
        . . .
} sem_t; /*opaque POSIX.4 semaphore*/
```

MT-LEVEL

MT-Safe

DESCRIPTION

sem_wait( ) and **sem_trywait( )** are the functions by which a calling thread waits or proceeds depending upon the state of a semaphore. A synchronizing process can proceed only if the value of the semaphore it accesses is currently greater than **0.**

If at the time of a call to either **sem_wait( )** or **sem_trywait( ),** the value of *sem* is positive, these functions decrement the value of the semaphore, return immediately, and allow the calling process to continue.

If the semaphore's value is **0:**

> **sem_wait( )** blocks, awaiting the semaphore to be released by another process (or LWP or thread).
>
> **sem_trywait( )** fails, returning immediately.

RETURN VALUES

If at the time of a call to either **sem_wait( )** or **sem_trywait( ),** the value of *sem* is positive, these functions return **0** on success. If the call was unsuccessful, the state of the semaphore is unchanged, the calling function returns **-1,** and sets **errno** to indicate the error condition.

ERRORS

EAGAIN	The value of *sem* was **0** when **sem_trywait( )** was called.
EINVAL	*sem* does not refer to a valid semaphore.
EINTR	**sem_wait( )** was interrupted by a signal.
ENOSYS	**sem_wait( )** and **sem_trywait( )** are not supported by this implementation.
EDEADLK	A deadlock condition was detected; i.e., two separate processes are waiting for an available resource to be released via a semaphore "held" by the other process.

EXAMPLES

The customer waiting-line in a bank may be analogous to the synchronization scheme of a semaphore utilizing **sem_wait( )** and **sem_trywait( ):**

```
/* cc [ flag ... ] file ... –lposix4 –lthread [ library ... ] */
#include <errno.h>
#define TELLERS 10
sem_t bank_line;          /* semaphore */
int banking_hours(), deposit_withdrawal;
void *customer(), do_business(), skip_banking_today();
thread_t tid;
...

sem_init(&bank_line,TRUE,TELLERS);/* 10 tellers available */
while(banking_hours())
        thr_create(NULL, NULL, customer, (void *)deposit_withdrawal,
                        THREAD_NEW_LWP, &tid);
...

void *
customer(deposit_withdrawal)
void *deposit_withdrawal;
{
        int this_customer, in_a_hurry = 50;
        this_customer = rand() % 100;
        if (this_customer == in_a_hurry) {
                if (sem_trywait(&bank_line) != 0)
                 if (errno == EAGAIN) { /* no teller available */
                                skip_banking_today(this_customer);
                                return;
                 } /*else go immediately to available teller & decrement bank_line*/
        }
        else
                sem_wait(&bank_line); /* wait for next teller, then proceed,
                                        and decrement bank_line */


        do_business((int *)deposit_withdrawal);
        sem_post(&bank_line); /* increment bank_line;
                                this_customer's teller
                                is now available */
}
```

SEE ALSO

sem_post(3R)

NOTES

sem_wait( ) can be interrupted by a signal, which may result in its premature return.

sem_post(3R) increments the semaphore upon its successful return.

BUGS

In Solaris 2.5, these functions always return –**1** and set **errno** to **ENOSYS,** because this release does not support the Semaphores option. It is our intention to provide support for these interfaces in future releases.