NAME

Thread::Semaphore - Thread-safe semaphores

VERSION

This document describes Thread::Semaphore version 2.13

SYNOPSIS

```
use Thread::Semaphore;
my $s = Thread::Semaphore->new();
$s->down();   # Also known as the semaphore P operation.
# The guarded section is here
$s->up();     # Also known as the semaphore V operation.

# Decrement the semaphore only if it would immediately succeed.
if ($s->down_nb()) {
    # The guarded section is here
    $s->up();
}

# Forcefully decrement the semaphore even if its count goes below 0.
$s->down_force();

# The default value for semaphore operations is 1
my $s = Thread::Semaphore->new($initial_value);
$s->down($down_value);
$s->up($up_value);
if ($s->down_nb($down_value)) {
    ...
    $s->up($up_value);
}
$s->down_force($down_value);
```

DESCRIPTION

Semaphores provide a mechanism to regulate access to resources. Unlike locks, semaphores aren't tied to particular scalars, and so may be used to control access to anything you care to use them for.

Semaphores don't limit their values to zero and one, so they can be used to control access to some resource that there may be more than one of (e.g., filehandles). Increment and decrement amounts aren't fixed at one either, so threads can reserve or return multiple resources at once.

METHODS

->new()

->new(NUMBER)

new creates a new semaphore, and initializes its count to the specified number (which must be an integer). If no number is specified, the semaphore's count defaults to 1.

->down()

->down(NUMBER)

The down method decreases the semaphore's count by the specified number (which must be an integer >= 1), or by one if no number is specified.

If the semaphore's count would drop below zero, this method will block until such time as the semaphore's count is greater than or equal to the amount you're downing the semaphore's count by.

This is the semaphore "P operation" (the name derives from the Dutch word "pak", which means "capture" -- the semaphore operations were named by the late Dijkstra, who was Dutch).

->down_nb()

->down_nb(NUMBER)

The `down_nb` method attempts to decrease the semaphore's count by the specified number (which must be an integer >= 1), or by one if no number is specified.

If the semaphore's count would drop below zero, this method will return *false*, and the semaphore's count remains unchanged. Otherwise, the semaphore's count is decremented and this method returns *true*.

->down_force()

->down_force(NUMBER)

The `down_force` method decreases the semaphore's count by the specified number (which must be an integer >= 1), or by one if no number is specified. This method does not block, and may cause the semaphore's count to drop below zero.

->down_timed(TIMEOUT)

->down_timed(TIMEOUT, NUMBER)

The `down_timed` method attempts to decrease the semaphore's count by 1 or by the specified number within the specified timeout period given in seconds (which must be an integer >= 0).

If the semaphore's count would drop below zero, this method will block until either the semaphore's count is greater than or equal to the amount you're `down`ing the semaphore's count by, or until the timeout is reached.

If the timeout is reached, this method will return *false*, and the semaphore's count remains unchanged. Otherwise, the semaphore's count is decremented and this method returns *true*.

->up()

->up(NUMBER)

The `up` method increases the semaphore's count by the number specified (which must be an integer >= 1), or by one if no number is specified.

This will unblock any thread that is blocked trying to `down` the semaphore if the `up` raises the semaphore's count above the amount that the `down` is trying to decrement it by. For example, if three threads are blocked trying to `down` a semaphore by one, and another thread `up`s the semaphore by two, then two of the blocked threads (which two is indeterminate) will become unblocked.

This is the semaphore "V operation" (the name derives from the Dutch word "vrij", which means "release").

NOTES

Semaphores created by *Thread::Semaphore* can be used in both threaded and non-threaded applications. This allows you to write modules and packages that potentially make use of semaphores, and that will function in either environment.

SEE ALSO

Thread::Semaphore on MetaCPAN: *https://metacpan.org/release/Thread-Semaphore*

Code repository for CPAN distribution: *https://github.com/Dual-Life/Thread-Semaphore*

threads, *threads::shared*

Sample code in the *examples* directory of this distribution on CPAN.

MAINTAINER

Jerry D. Hedden, <jdhedden AT cpan DOT org>

LICENSE