class Semaphore::ReadersWriters

Semaphore readers writers pattern

Table of Contents

```
class Semaphore::ReadersWriters { ... }
```

Synopsis

```
use Semaphore::ReadersWriters;

my Semaphore::ReadersWriters $rw .= new;
$rw.add-mutex-names('shv');
my $shared-var = 10;

# After creating threads ...
# Some writer thread
$rw.writer( 'shv', {$shared-var += 2});

# Some reader thread
say 'Shared var is ', $rw.reader( 'shv', {$shared-var;});
```

Description

This readers writers pattern is a pattern describing the behavior of types of threads using shared data. In this description it is allowed to have more than one thread accessing the shared data for reading and only one thread for writing. This means that when there are readers in the critical area the writers must wait until all readers left this area. The critical area is a piece of code where the shared data is manipulated. Readers and writers have to wait when there is a writer in that area. There are several solutions to this pattern. This class provides three of them;

- *Readers priority*. When readers are in the critical area, the writer has to wait until all readers have left. When new readers arive before the last reader leaves the area, the area stays occupied and can lead to writers starvation where the data in the critical section never gets updated.

- *No writer starvation*. This method will block new readers as soon as there is a writer. After the writer leaves the critical area, the next reader or writer can access the area.

- *Writer priority*. In the second method, it is possible that a waiting reader is entered first while there might be another writer waiting. It would be better for the reader to have to wait until all writers have finished writing and read the most up to date data. A possible problem would now be that there can be a reader starvation situation.

Methods

add-mutex-names

Defined as

```
method add-mutex-names (
  *@snames,
  RWPatternType :$RWPatternType = C-RW-WRITERPRIO
)
```

With this class it is possible to control several shared areas and uses names to separate them. This method sets one or more names. With the name a variation to the reader writer theme can be specified in :RWPatternType. The method throws an exception when a key in the @snames array is already in use.

- C-RW-READERPRIO. Readers have priority, see description above.

- C-RW-NOWRITERSTARVE. No writer starvation.

- C-RW-WRITERPRIO, Writers have priority. This is the default.

rm-mutex-names

Defined as

```
method rm-mutex-names (*@snames)
```

Method to remove the name and its structures.

get-mutex-names ( )

Defined as

```
method get-mutex-names ()
```

Return all defined names.

check-mutex-names

Defined as

```
method check-mutex-names ( *@names --> Bool )
```

Check if a name from @names is in use. Returns True if so, False otherwise.

reader

Defined as

```
method reader ( Str:D $sname, Block $code)
```

Method to access the shared data using the provided $code. When the $sname is not defined before with add-mutex-names, the method throws an exception. There are no checks to make sure that the shared data is really only read and not modified by the provided code.

Example

```
my @data = 1..^5; # protected variable
my Semaphore::ReadersWriters $rw .= new;
$rw.add-mutex-names( 'data', :RWPatternType(C-RW-WRITERPRIO));

# Later in a thread ...
say "Xyz is ", $rw.reader( 'data', {
    my $xyz = 10;
    $xyz *= $_ for @data;
    $xyz;
  }
);
```

writer

Defined as

```
method writer ( Str:D $sname, Block $code)
```

Method to access the shared data using the provided $code. When the $sname is not defined before with add-mutex-names, the method throws an exception.

Example

```
my @data = 1..^5; # protected variable
my Semaphore::ReadersWriters $rw .= new;
$rw.add-mutex-names( 'data', :RWPatternType(C-RW-WRITERPRIO));

# Later in a thread ...
say "sum of data is ", $rw.writer( 'data', {
    for @data -> $d is rw { $d += 2; }
    [+] @data;
  }
);
```