

SSC Addendum 2 (proposal)

Read/Write Locks Fairness

This document is an Addendum to the Shared State Concurrency specification. It is dedicated to the public domain.

Rationale

TD: Write the rationale.

Specification

A read/write lock has a **fairness property**, which is its behavior when there are reader and writer threads contending to lock the read/write lock. A **fair** or **unbiased** lock is one in which readers and writers have access to the lock in the order that they ask for it: readers wait for earlier writers; writers wait for earlier readers and writers. A **reader biased** lock is one in which preference is given to the readers, allowing a new readers to join a group of current readers even if there are writers waiting. A reader biased lock minimizes the delay for readers at the expense of readers reading possible outdated data. A **writer biased** lock is one in which preference is given to the writers, making new readers wait for any current or waiting writer. A writer biased lock ensures that updates are seen as soon as possible at the expense of less throughput of readers.

This adds a *fairness* parameter to the function `make-rwlock`.

`make-rwlock &key (fairness :fair) &allow-other-keys` [Function]

The argument *fairness* can be one of `:fair`, `:read-bias`, or `:write-bias`. It hints if the lock should be fair, reader biased or writer biased. It is an error if *fairness* is any other object. The default is `:fair`.

`rwlock-property rwlock :fairness` [Function]

This returns the fairness property of *rwlock*. It returns one of `:fair`, `:read-bias`, or `:write-bias`.

`setf (rwlock-property rwlock :fairness) fairness` [Function]

This sets the fairness property of *rwlock* to *fairness*, which can be one of `:fair`, `:read-bias`, or `:write-bias`. *rwlock* must be an read/write lock.

Depending on the implementation, it might not be easy to change the fairness property after the read/write lock is created. It is perfectly legal for the implementation to ignore attempts to change it. When this happens, the implementation must report

```
(defvar *l* (make-rwlock :fairness :read-bias))
(rwlock-property *l* :fairness)
(setf (rwlock-property *l* :fairness) :write-bias)
(rwlock-property *l* :fairness)
```

The user is urged to choose an appropriate default for this property when creating the read/write lock, as attempts to change it may be ignored.

An implementation may not implement all three kinds of 'fairness'. For example, it may provide an algorithm for reader biased and another for both fair and writer biased.