Unstable-Funcons-beta: Locks *

The PLanCompS Project

`Locks.cbs` | PLAIN | PRETTY

OUTLINE

Locks

Locks

[*Funcon* is-exclusive-lock-holder

Spin locks

Funcon spin-lock-create

Funcon spin-lock-sync

Funcon spin-lock-release

Exclusive locks

Funcon exclusive-lock-create

Funcon exclusive-lock-sync

Funcon exclusive-lock-sync-else-wait

Funcon exclusive-lock-release

Reentrant locks

Funcon reentrant-lock-create

Funcon reentrant-lock-sync

Funcon reentrant-lock-sync-else-wait

Funcon reentrant-lock-release

Funcon reentrant-lock-exit

Semaphores

Funcon semaphore-create

Funcon semaphore-sync

Funcon semaphore-sync-else-wait

Funcon semaphore-release

Shared-exclusive locks

Funcon rw-lock-create

Funcon rw-lock-sync-exclusive

Funcon rw-lock-sync-shared

Funcon rw-lock-sync-exclusive-else-wait

Funcon rw-lock-sync-shared-else-wait

Funcon rw-lock-release-exclusive

Funcon rw-lock-release-shared]

A thread may request locks, and release locks (held by itself or by another thread). A reentrant lock may be held more than once by the same thread. A shared lock may be held by multiple threads at

*Suggestions for improvement: `plancomps@gmail.com`.
Reports of issues: `https://github.com/plancomps/CBS-beta/issues`.

the same time, whereas an exclusive lock can be held by only one thread at the same time. A request for a spinlock that is held by another thread involves busy waiting instead of suspension, and assumes interleaving of the execution of a waiting thread and the holding thread.

Funcon is-exclusive-lock-holder(*SY* : syncs) : ⇒ booleans
⇝ is-equal(current-thread, assigned sync-feature(*SY*, sync-holder))

Spin locks Spin locks support mutual exclusion, but not suspension of blocked requests. thread-spin spin-lock-sync *SY* repeatedly executes the request for the lock until the request is granted, which is called busy waiting. Interleaving of different threads waiting for the same spin lock may result in granting requests out of order.

Funcon spin-lock-create : ⇒ syncs
⇝ sync-create(
sync-feature-create sync-held,
sync-feature-create sync-holder)

When the lock is not currently held, granting a request for it sets the holder to the current thread; otherwise the request fails.

Funcon spin-lock-sync(*SY* : syncs) : ⇒ null-type
⇝ thread-atomic sequential(
check-true not(assigned sync-feature(*SY*, sync-held)),
assign(sync-feature(*SY*, sync-held), true),
assign(sync-feature(*SY*, sync-holder), current-thread))

Releasing the lock leaves the lock free. Only the thread that holds the lock can release it. Releasing cannot be blocked by other threads, so it is not a request.

Funcon spin-lock-release(*SY* : syncs) : ⇒ null-type
⇝ thread-atomic sequential(
check-true is-exclusive-lock-holder(*SY*),
assign(sync-feature(*SY*, sync-held), false),
un-assign(sync-feature(*SY*, sync-holder)))

Exclusive locks Exclusive locks support mutual exclusion and suspension of blocked requests. An exclusive lock – also called a mutex – can be held by only one thread at the same time. It can be used to ensure mutual exclusion of so-called critical regions of thread bodies, and to avoid potential interference due to thread interleaving.

Funcon exclusive-lock-create : ⇒ syncs
⇝ sync-create(
sync-feature-create sync-waiting-list,
sync-feature-create sync-held,
sync-feature-create sync-holder)

When the lock is not currently held, granting a request for it sets the holder to the current thread; otherwise the request fails.

```
Funcon  exclusive-lock-sync(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic sequential(
                check-true not(assigned sync-feature(SY, sync-held)),
                assign(sync-feature(SY, sync-held), true),
                assign(sync-feature(SY, sync-holder), current-thread))
```

When the request fails, the current thread is added to the waiting list, and suspended until the request can be granted:

```
Funcon  exclusive-lock-sync-else-wait(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic else(
                exclusive-lock-sync(SY),
                sequential(
                  sync-waiting-list-add(SY, current-thread),
                  thread-suspend current-thread))
```

When the waiting list is non-empty, releasing the lock grants it to the thread that made the first request in the list, and resumes that thread; otherwise it leaves the lock free. Only the thread that holds the lock can release it. Releasing a lock cannot be blocked by other threads, so it is not a request.

```
Funcon  exclusive-lock-release(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic sequential(
                check-true is-exclusive-lock-holder(SY),
                if-true-else(
                  is-equal(assigned sync-feature(SY, sync-waiting-list), [ ]),
                  sequential(
                    assign(sync-feature(SY, sync-held), false),
                    un-assign(sync-feature(SY, sync-holder))),
                  give(
                    sync-waiting-list-head-remove(SY),
                    sequential(
                      assign(sync-feature(SY, sync-holder), given),
                      thread-resume given))))
```

Rentrant locks Reentrant exclusive locks support mutual exclusion, suspension of blocked requests, and reentry. A reentrant exclusive lock can be held (and subsequently released) multiple times by the same thread.

```
Funcon  reentrant-lock-create : ⇒ syncs
          ⇝ sync-create(
                sync-feature-create sync-waiting-list,
                sync-feature-create sync-held,
                sync-feature-create sync-holder,
                sync-feature-create sync-count)
```

When the lock is not currently held, granting a request for it sets the holder to the current thread; if it is already held by the current thread, it merely increments the counter; otherwise the request fails.

```
Funcon  reentrant-lock-sync(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic else(
                sequential(
                  check-true not(assigned sync-feature(SY, sync-held)),
                  assign(sync-feature(SY, sync-held), true),
                  assign(sync-feature(SY, sync-holder), current-thread),
                  assign(sync-feature(SY, sync-count), 0)),
                sequential(
                  check-true assigned sync-feature(SY, sync-held),
                  check-true is-exclusive-lock-holder(SY),
                  assign(
                    sync-feature(SY, sync-count),
                    nat-succ assigned sync-feature(SY, sync-count))))
```

When the request fails, the current thread is added to the waiting list, and suspended until the request can be granted:

```
Funcon  reentrant-lock-sync-else-wait(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic else(
                reentrant-lock-sync(SY),
                sequential(
                  sync-waiting-list-add(SY, current-thread),
                  thread-suspend current-thread))
```

When the waiting list is non-empty, releasing the lock grants it to the thread that made the first request in the list, and resumes that thread; otherwise it leaves the lock free. Only the thread that holds the lock can release it. Releasing a lock cannot be blocked by other threads, so it is not a request.

```
Funcon  reentrant-lock-release(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic sequential(
                check-true is-exclusive-lock-holder(SY),
                if-true-else(
                  is-equal(assigned sync-feature(SY, sync-waiting-list), [ ]),
                  sequential(
                    assign(sync-feature(SY, sync-held), false),
                    un-assign(sync-feature(SY, sync-holder)),
                    assign(sync-feature(SY, sync-count), 0)),
                  give(
                    sync-waiting-list-head-remove(SY),
                    sequential(
                      assign(sync-feature(SY, sync-holder), given),
                      assign(sync-feature(SY, sync-count), 0),
                      thread-resume given))))
```

When the reentered count is positive, an exit merely decrements it. Otherwise it is 0, and the exit releases the lock.

```
Funcon  reentrant-lock-exit(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic sequential(
                check-true is-exclusive-lock-holder(SY),
                give(
                  sync-feature(SY, sync-count),
                  if-true-else(
                    is-greater(assigned given, 0),
                    assign(given, checked nat-pred assigned given),
                    reentrant-lock-release(SY))))
```

Semaphores A semaphore is a shared lock with a specified limit on the number of threads that can hold it at the same time. A semaphore can be released by any thread.

```
Funcon  semaphore-create(N : pos-ints) : ⇒ syncs
          ⇝ give(
                sync-create(
                  sync-feature-create sync-waiting-list,
                  sync-feature-create sync-count),
                sequential(
                  assign(sync-feature(given, sync-count), N),
                  given))
```

When the semaphore is available, granting a request for it decrements the number of further threads that can hold it; otherwise the request fails.

```
Funcon  semaphore-sync(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic sequential(
                check-true is-greater(assigned sync-feature(SY, sync-count), 0),
                assign(
                  sync-feature(SY, sync-count),
                  checked nat-pred assigned sync-feature(SY, sync-count)))
```

When the request fails, the current thread is added to the waiting list, and suspended until the request can be granted:

```
Funcon  semaphore-sync-else-wait(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic else(
                semaphore-sync(SY),
                sequential(
                  sync-waiting-list-add(SY, current-thread),
                  thread-suspend current-thread))
```

When the waiting list is empty, releasing the semaphore increments the counter; otherwise it grants the semaphore to the thread that made the first request in the list, and resumes that thread. Releasing a semaphore cannot be blocked, so it is not a request.

```
Funcon  semaphore-release(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic
              if-true-else(
                is-equal(assigned sync-feature(SY, sync-waiting-list), [ ]),
                assign(
                  sync-feature(SY, sync-count),
                  nat-succ assigned sync-feature(SY, sync-count)),
                give(
                  sync-waiting-list-head-remove(SY),
                  thread-resume given))
```

Shared-exclusive locks A shared-exclusive lock – also called a readers-writer (rw) lock – can be held exclusively by a single thread, or shared by any number of threads at the same time. It can be released by any thread.

```
Funcon  rw-lock-create : ⇒ syncs
          ⇝ give(
              sync-create(
                sync-feature-create sync-waiting-list,
                sync-feature-create sync-held,
                sync-feature-create sync-count),
              sequential(
                assign(sync-feature(given, sync-count), 0),
                given))
```

When the lock is not currently held at all, it can be granted exclusively:

```
Funcon  rw-lock-sync-exclusive(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic sequential(
              check-true and(
                not(assigned sync-feature(SY, sync-held)),
                is-equal(assigned sync-feature(SY, sync-count), 0)),
              assign(sync-feature(SY, sync-held), true))
```

When the lock is not currently held exclusively, a request to share it is always granted immediately (regardless of any waiting exclusive requests):

```
Funcon  rw-lock-sync-shared(SY : syncs) : ⇒ null-type
          ⇝ thread-atomic sequential(
              check-true not(assigned sync-feature(SY, sync-held)),
              assign(
                sync-feature(SY, sync-count),
                nat-succ assigned sync-feature(SY, sync-count)))
```

If the request fails, the current thread is added to the waiting list, and suspended until the request can be granted.

The waiting list of a shared-exclusive lock records not only the thread but also whether the request is for sharing:

Funcon rw-lock-sync-exclusive-else-wait(SY : syncs) : ⇒ null-type
⇝ thread-atomic else(
rw-lock-sync-exclusive(SY),
sequential(
sync-waiting-list-add(SY, tuple(current-thread, false)),
thread-suspend current-thread))

Funcon rw-lock-sync-shared-else-wait(SY : syncs) : ⇒ null-type
⇝ thread-atomic else(
rw-lock-sync-shared(SY),
sequential(
sync-waiting-list-add(SY, tuple(current-thread, true)),
thread-suspend current-thread))

When the waiting list is non-empty, releasing the lock may grant either the first waiting exclusive request, or all waiting shared requests. A scheduler may defer granting one kind of request when there are waiting requests of the other kind, irrespective of the order in which those requests were made. Releasing a lock cannot be blocked by other threads, so it is not a request.

Funcon rw-lock-release-exclusive(SY : syncs) : ⇒ null-type
⇝ thread-atomic sequential(
check-true assigned sync-feature(SY, sync-held),
assign(sync-feature(SY, sync-held), false),
rw-lock-sync(SY))

Funcon rw-lock-release-shared(SY : syncs) : ⇒ null-type
⇝ thread-atomic sequential(
assign(
sync-feature(SY, sync-count),
checked nat-pred assigned sync-feature(SY, sync-count)),
if-true-else(
is-equal(0, assigned sync-feature(SY, sync-count)),
rw-lock-sync(SY),
null-value))

rw-lock-sync(SY) assumes that SY is not held (either exclusively or shared). If the first waiting request is for sharing, any further sharing requests are granted,

```
Auxiliary Funcon  rw-lock-sync(SY : syncs) : ⇒ null-type
                    ⇝ if-true-else(
                          is-equal(assigned sync-feature(SY, sync-waiting-list), [ ]),
                          null-value,
                          give(
                            sync-waiting-list-head-remove(SY),
                            sequential(
                              thread-resume first tuple-elements given,
                              if-true-else(
                                second tuple-elements given,
                                sequential(
                                  assign(
                                    sync-feature(SY, sync-count),
                                    nat-succ assigned sync-feature(SY, sync-count)),
                                  rw-lock-sync-all-shared(SY)),
                                assign(sync-feature(SY, sync-held), true)))))
```

rw-lock-sync-all-shared(*SY*) updates the waiting list by removing and resuming all its sharing requests:

```
Auxiliary Funcon  rw-lock-sync-all-shared(SY : syncs) : ⇒ null-value
                    ⇝ assign(
                          sync-feature(SY, sync-waiting-list),
                          [
                          left-to-right-filter(
                            if-true-else(
                              second tuple-elements given,
                              sequential(
                                thread-resume first tuple-elements given,
                                assign(
                                  sync-feature(SY, sync-count),
                                  nat-succ assigned sync-feature(SY, sync-count)),
                                false),
                              true),
                            list-elements assigned sync-feature(SY, sync-waiting-list))])
```