NAME

"Future::Mutex" − mutual exclusion lock around code that returns Futures

SYNOPSIS

```
use Future::Mutex;

my $mutex = Future::Mutex->new;

sub do_atomically
{
   return $mutex->enter( sub {
      ...
      return $f;
   });
}
```

DESCRIPTION

Most Future−using code expects to run with some level of concurrency, using future instances to represent still-pending operations that will complete at some later time. There are occasions however, when this concurrency needs to be restricted − some operations that, once started, must not be interrupted until they are complete. Subsequent requests to perform the same operation while one is still outstanding must therefore be queued to wait until the first is finished. These situations call for a mutual-exclusion lock, or "mutex".

A `Future::Mutex` instance provides one basic operation, which will execute a given block of code which returns a future, and itself returns a future to represent that. The mutex can be in one of two states; either unlocked or locked. While it is unlocked, requests to execute code are handled immediately. Once a block of code is invoked, the mutex is now considered to be locked, causing any subsequent requests to invoke code to be queued behind the first one, until it completes. Once the initial code indicates completion (by its returned future providing a result or failing), the next queued code is invoked.

An instance may also be a counting mutex if initialised with a count greater than one. In this case, it can keep multiple blocks outstanding up to that limit, with subsequent requests queued as before. This allows it to act as a concurrency-bounding limit around some operation that can run concurrently, but an application wishes to apply overall limits to stop it growing too much, such as communications with external services or executing other programs.

CONSTRUCTOR

new

```
$mutex = Future::Mutex->new( count => $n )
```

Returns a new `Future::Mutex` instance. It is initially unlocked.

Takes the following named arguments:

count => INT

Optional number to limit outstanding concurrency. Will default to 1 if not supplied.

METHODS

enter

```
$f = $mutex->enter( \&code )
```

Returns a new `Future` that represents the eventual result of calling the code. If the mutex is currently unlocked, the code will be invoked immediately. If it is currently locked, the code will be queued waiting for the next time it becomes unlocked.

The code is invoked with no arguments, and is expected to return a `Future`. The eventual result of that future determines the result of the future that `enter` returned.

available

```
$avail = $mutex->available
```

Returns true if the mutex is currently unlocked, or false if it is locked.

AUTHOR

Paul Evans <leonerd@leonerd.org.uk>