Package 'flock'

October 13, 2022

Type Package

Title Process Synchronization Using File Locks

Version 0.7

Date 2016-11-10

Author Ivan Popivanov

Maintainer Ivan Popivanov <ivan.popivanov@gmail.com>

Description Implements synchronization between R processes (spawned by using the ``parallel'' package for instance) using file locks. Supports both exclusive and shared locking.

License Apache License 2.0

Imports Rcpp (>= 0.11.3), methods

LinkingTo Rcpp

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-11-12 16:23:32

R topics documented:

---

flock-package   *Process synchronization using file locks.*

Description

Enables synchronization between R processes using a file lock. Supports both exclusive (writer) and shared (readers) locks.

On UNIX, the implementation relies on the `fcntl` system call. While on Windows, the `LockFileEx/UnlockFileEx` APIs are used.

Details

Package:	flock
Type:	Package
Version:	0.6
Date:	2014-11-24
License:	Apache License 2.0

Author(s)

Ivan Popivanov

Maintainer: Ivan Popivanov <ivan.popivanov@gmail.com>

Examples

```
## Not run:
require(DBI)
require(flock)
require(RSQLite)
require(parallel)

dbpath <- tempfile()
con <- dbConnect(RSQLite::SQLite(), dbname=dbpath)
df <- data.frame(value = 0)
dbWriteTable(con, "test", df)
dbDisconnect(con)

write.one.value <- function(val, lock.name=NULL) {
   if(!is.null(lock.name)) {
      file.lock = lock(lock.name)
   }

   # The three lines below are the "critical section"
   con <- dbConnect(RSQLite::SQLite(), dbname = dbpath)
```

```
      dbWriteTable(con, "test", data.frame(value = val), append = TRUE)
      dbDisconnect(con)

      if(!is.null(lock.name)) {
         unlock(file.lock)
      }
   }

   lock.name = tempfile()

   # Run the parallel database updates with two cores
   mclapply(1:100, write.one.value, mc.cores=2, lock.name=lock.name)

   # To see the failing scenario, run (on a multi-core system):
   # mclapply(1:100, write.one.value, mc.cores=2)

   ## End(Not run)
```

---

is.locked	*Locking/Unlocking*

Description

Checks whether a lock has been obtained.

Usage

```
is.locked(file.lock)
```

Arguments

file.lock   The lock as an object of type `FileLock`.

Author(s)

Ivan Popivanov

Examples

```
## Not run:
require(flock)

file.lock = lock("~/file.lock")
# Critical section code goes between here and the unlock call
if(is.locked(file.lock)) {
   print("Got the lock!")
}
unlock(file.lock)

## End(Not run)
```

---

lock   *Locking/Unlocking*

Description

Locks a file in exclusive or shared mode.

Usage

```
lock(path, exclusive = TRUE)
```

Arguments

path   Character. The path.

exclusive   Logical. The lock type, exclusive or shared.

Details

The file is created if it doesn't exist.

Value

Returns an object of type `FileLock`, which is to be used for the `unlock` call.

Author(s)

Ivan Popivanov

Examples

```
## Not run:
require(flock)

file.lock = lock("~/file.lock")
# Critical section code goes here
unlock(file.lock)

## End(Not run)
```

---

unlock   *Locking/Unlocking*

Description

Unlocks a file previously locked via lock.

Usage

```
unlock(file.lock)
```

Arguments

file.lock   The FileLock object returned by lock.

Author(s)

Ivan Popivanov

See Also

lock

Examples

```
## Not run:
require(flock)

file.lock = lock("~/file.lock")
# Critical section code goes here
unlock(file.lock)

## End(Not run)
```

Index