Package ‘filelock’

December 11, 2023

Title Portable File Locking

Version 1.0.3

Description Place an exclusive or shared lock on a file. It uses 'LockFile' on Windows and 'fcntl' locks on Unix-like systems.

License MIT + file LICENSE

URL `https://r-lib.github.io/filelock/`, `https://github.com/r-lib/filelock`

BugReports `https://github.com/r-lib/filelock/issues`

Depends R (>= 3.4)

Suggests callr (>= 2.0.0), covr, testthat (>= 3.0.0)

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation yes

Author Gábor Csárdi [aut, cre], Posit Software, PBC [cph, fnd]

Maintainer Gábor Csárdi <`csardi.gabor@gmail.com`>

Repository CRAN

Date/Publication 2023-12-11 04:40:02 UTC

R topics documented:

lock — *Advisory File Locking and Unlocking*

Description

There are two kinds of locks, *exclusive* and *shared*, see the `exclusive` argument and other details below.

Usage

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

unlock(lock)
```

Arguments

`path`	Path to the file to lock. If the file does not exist, it will be created, but the directory of the file must exist. *Do not place the lock on a file that you want to read from or write to!* *Always use a special lock file. See details below.
`exclusive`	Whether to acquire an exclusive lock. An exclusive lock gives the process exclusive access to the file, no other processes can place any kind of lock on it. A non-exclusive lock is a shared lock. Multiple processes can hold a shared lock on the same file. A process that writes to a file typically requests an exclusive lock, and a process that reads from it typically requests a shared lock.
`timeout`	Timeout to acquire the lock in milliseconds. If `Inf`, then the process will wait indefinitely to acquire the lock. If zero, then the function it returns immediately, with or without acquiring the lock
`lock`	The lock object to unlock. It is not an error to try to unlock an already unlocked lock. It is not possible to lock an unlocked lock again, a new lock has to be requested.

Value

`lock` returns a `filelock_lock` object if the lock was successfully acquired, and `NULL` if a timeout happened.

`unlock` returns `TRUE`, always.

Warning

Always use special files for locking. I.e. if you want to restrict access to a certain file, do *not* place the lock on this file. Create a special file, e.g. by appending `.lock` to the original file name and place the lock on that. (The `lock()` function creates the file for you, actually, if it does not exist.) Reading from or writing to a locked file has undefined behavior! (See more about this below at the Internals Section.)

It is hard to determine whether and when it is safe to remove these special files, so our current recommendation is just to leave them around.

It is best to leave the special lock file empty, simply because on some OSes you cannot write to it (or read from it), once the lock is in place.

Advisory Locks

All locks set by this package might be advisory. A process that does not respect this locking mechanism may be able to read and write the locked file, or even remove it (assuming it has capabilities to do so).

Unlock on Termination

If a process terminates (with a normal exit, a crash or on a signal), the lock(s) it is holding are automatically released.

If the R object that represents the lock (the return value of `lock`) goes out of scope, then the lock will be released automatically as soon as the object is garbage collected. This is more of a safety mechanism, and the user should still `unlock()` locks manually, maybe using `base::on.exit()`, so that the lock is released in case of errors as well, as soon as possible.

Special File Systems

File locking needs support from the file system, and some *non-standard* file systems do not support it. For example on network file systems like NFS or CIFS, user mode file systems like `sshfs` or `ftpfs`, etc., support might vary. Recent Linux versions and recent NFS versions (from version 3) do support file locking, if enabled.

In theory it is possible to simply test for lock support, using two child processes and a timeout, but `filelock` does not do this currently.

Locking Part of a File

While this is possible in general, `filelock` does not support it currently. The main purpose of `filelock` is to lock using special lock files, and locking part of these is not really useful.

Internals on Unix

On Unix (i.e. Linux, macOS, etc.), we use `fcntl` to acquire and release the locks. You can read more about it here: `https://www.gnu.org/software/libc/manual/html_node/File-Locks.html`

Some important points:

- The lock is put on a file descriptor, which is kept open, until the lock is released.
- A process can only have one kind of lock set for a given file.
- When any file descriptor for that file is closed by the process, all of the locks that process holds on that file are released, even if the locks were made using other descriptors that remain open. Note that in R, using a one-shot function call to modify the file opens and closes a file descriptor to it, so the lock will be released. (This is one of the main reasons for using special lock files, instead of putting the lock on the actual file.)
- Locks are not inherited by child processes created using fork.

- For lock requests with finite timeout intervals, we set an alarm, and temporarily install a signal handler for it. R is single threaded, so no other code can be running, while the process is waiting to acquire the lock. The signal handler is restored to its original value immediately after the lock is acquired or the timeout expires. (It is actually restored from the signal handler, so there should be no race conditions here. However, if multiple `SIGALRM` signals are delivered via a single call to the signal handler, then alarms might get lost. Currently base R does not use the `SIGALRM` signal for anything, but other packages might.)

Internals on Windows

On Windows, `LockFileEx` is used to create the lock on the file. If a finite timeout is specified for the lock request, asynchronous (overlapped) I/O is used to wait for the locking event with a timeout. See more about `LockFileEx` on the first hit here: `https://www.google.com/search?q=LockFileEx`

Some important points:

- `LockFileEx` locks are mandatory (as opposed to advisory), so indeed no other processes have access to the locked file. Actually, even the locking process has no access to it through a different file handle, than the one used for locking. In general, R cannot read from the locked file, and cannot write to it. (Although, the current R version does not fail, it just does nothing, which is quite puzzling.) Remember, always use a special lock file, instead of putting the lock on the main file, so that you are not affected by these problems.
- Inherited handles do not provide access to the child process.

Examples

```
## -------------------------------------------------------------
## R process 1 gets an exclusive lock
## Warning: if you want to lock file 'myfile', always create a
## separate lock file instead of placing the lock on this file directly!
lck <- lock(mylockfile)

## -------------------------------------------------------------
## R process 2 fails to acquire a lock
lock(mylockfile, timeout = 0)

## Let's wait for 5 seconds, before giving up
lock(mylockfile, timeout = 5000)

## Wait indefinetely
lock(mylockfile, timeout = Inf)
```

Index