Package 'interprocess'

May 23, 2025

Type Package

Title Mutexes, Semaphores, and Message Queues

Version 1.3.0

Date 2025-05-23

Description Provides access to low-level operating system mechanisms for performing atomic operations on shared data structures. Mutexes provide shared and exclusive locks. Semaphores act as counters. Message queues move text strings from one process to another. All these interprocess communication (IPC) tools can optionally block with or without a timeout. Implemented using the cross-platform 'boost' 'C++' library <https://www.boost.org/doc/libs/release/libs/interprocess/>.

URL https://cmmr.github.io/interprocess/, https://github.com/cmmr/interprocess

BugReports https://github.com/cmmr/interprocess/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

Config/Needs/website rmarkdown

LinkingTo cpp11, BH

Suggests callr, testthat

NeedsCompilation yes

Author Daniel P. Smith [aut, cre] (ORCID: <https://orcid.org/0000-0002-2479-2044>), Alkek Center for Metagenomics and Microbiome Research [cph, fnd]

Maintainer Daniel P. Smith <dansmith01@gmail.com>

Repository CRAN

Date/Publication 2025-05-23 17:52:05 UTC

Contents

`msg_queue` *Send Text Messages Between Processes*

Description

An interprocess message queue that ensures each message is delivered to only one reader, at which time the message is removed from the queue. Ideal for producer/consumer situations where the message defines work waiting to be processed. The message itself can be any scalar character, for example, a JSON string or path to an RDS file.

Usage

```
msg_queue(
  name = uid(),
  assert = NULL,
  max_count = 100,
  max_nchar = 128,
  cleanup = FALSE,
  file = NULL
)

## S3 method for class 'msg_queue'
with(data, expr, alt_expr = NULL, timeout_ms = Inf, ...)
```

Arguments

`name`	Unique ID. Alphanumeric, starting with a letter.
`assert`	Apply an additional constraint. • `'create'` - Error if the message queue **already exists**. • `'exists'` - Error if the message queue **doesn't exist**. • `NULL` - No constraint; create the message queue if it doesn't exist.
`max_count`	The maximum number of messages that can be stored in the queue at the same time. Attempting to send additional messages will cause `send()` to block or return `FALSE`. Ignored if the message queue already exists.
`max_nchar`	The maximum number of characters in each message. Attempting to send larger messages will throw an error. Ignored if the message queue already exists.
`cleanup`	Remove the message queue when the R session exits. If `FALSE`, the message queue will persist until `$remove()` is called or the operating system is restarted.

file	Use a hash of this file/directory path as the message queue name. The file itself will not be read or modified, and does not need to exist.
data	A msg_queue object.
expr	Expression to evaluate if a message is received. The message can be accessed by . in this context. See examples.
alt_expr	Expression to evaluate if timeout_ms is reached.
timeout_ms	Maximum time (in milliseconds) to block the process while waiting for the operation to succeed. Use 0 or Inf to return immediately or only when successful, respectively.
...	Not used.

Value

msg_queue() returns a msg_queue object with the following methods:

- $name
  - Returns the message queue's name (scalar character).
- $send(msg, timeout_ms = Inf, priority = 0)
  - Returns TRUE on success, or FALSE if the timeout is reached.
  - msg: The message (scalar character) to add to the message queue.
  - priority: Higher priority messages will be retrieved from the message queue first. 0 = lowest priority; integers only.
- $receive(timeout_ms = Inf)
  - Returns the next message from the message queue, or NULL if the timeout is reached.
- $count()
  - Returns the number of messages currently in the message queue.
- $max_count()
  - Returns the maximum number of messages the queue can hold.
- $max_nchar()
  - Returns the maximum number of characters per message.
- $remove()
  - Returns TRUE if the message queue was successfully deleted from the operating system, or FALSE on error.

with() returns eval(expr) on success; eval(alt_expr) otherwise.

See Also

Other shared objects: mutex(), semaphore()

Examples

```
mq <- interprocess::msg_queue()
print(mq)

mq$send(paste('my favorite number is', floor(runif(1) * 100)))
mq$count()

mq$receive()
mq$receive(timeout_ms = 0)

mq$send('The Matrix has you...')
with(mq, paste('got message:', .), 'no messages', timeout_ms = 0)
with(mq, paste('got message:', .), 'no messages', timeout_ms = 0)

mq$remove()
```

`mutex` *Shared and Exclusive Locks*

Description

Mutually exclusive (mutex) locks are used to control access to shared resources.

An *exclusive lock* grants permission to one process at a time, for example to update the contents of a database file. While an exclusive lock is active, no other exclusive or shared locks will be granted.

Multiple *shared locks* can be held by different processes at the same time, for example to read a database file. While a shared lock is active, no exclusive locks will be granted.

Usage

```
mutex(name = uid(), assert = NULL, cleanup = FALSE, file = NULL)

## S3 method for class 'mutex'
with(data, expr, alt_expr = NULL, shared = FALSE, timeout_ms = Inf, ...)
```

Arguments

`name`	Unique ID. Alphanumeric, starting with a letter.
`assert`	Apply an additional constraint. • `'create'` - Error if the mutex *already exists*. • `'exists'` - Error if the mutex *doesn't exist*. • `NULL` - No constraint; create the mutex if it doesn't exist.
`cleanup`	Remove the mutex when the R session exits. If `FALSE`, the mutex will persist until `$remove()` is called or the operating system is restarted.
`file`	Use a hash of this file/directory path as the mutex name. The file itself will not be read or modified, and does not need to exist.

data	A mutex object.
expr	Expression to evaluate if the mutex is acquired.
alt_expr	Expression to evaluate if timeout_ms is reached.
shared	If FALSE (the default) an exclusive lock is returned. If TRUE, a shared lock is returned instead. See description.
timeout_ms	Maximum time (in milliseconds) to block the process while waiting for the operation to succeed. Use 0 or Inf to return immediately or only when successful, respectively.
...	Not used.

Details

The operating system ensures that mutex locks are released when a process exits.

Value

mutex() returns a mutex object with the following methods:

- $name
  - Returns the mutex's name (scalar character).
- $lock(shared = FALSE, timeout_ms = Inf)
  - Returns TRUE if the lock is acquired, or FALSE if the timeout is reached.
- $unlock(warn = TRUE)
  - Returns TRUE if successful, or FALSE (with optional warning) if the mutex wasn't locked by this process.
- $remove()
  - Returns TRUE if the mutex was successfully deleted from the operating system, or FALSE on error.

with() returns eval(expr) if the lock was acquired, or eval(alt_expr) if the timeout is reached.

Error Handling

The with() wrapper automatically unlocks the mutex if an error stops evaluation of expr. If you are directly calling lock(), be sure that unlock() is registered with error handlers or added to on.exit(). Otherwise, the lock will persist until the process terminates.

Duplicate Mutexes

Mutex locks are per-process. If a process already has a lock, it can not attempt to acquire a second lock on the same mutex.

See Also

Other shared objects: msg_queue(), semaphore()

Examples

```
tmp <- tempfile()
mut <- interprocess::mutex(file = tmp)

print(mut)

# Exclusive lock to write the file
with(mut, writeLines('some data', tmp))

# Use a shared lock to read the file
with(mut,
  shared     = TRUE,
  timeout_ms = 0,
  expr       = readLines(tmp),
  alt_expr   = warning('Mutex was locked. Giving up.') )

# Directly lock/unlock with safeguards
if (mut$lock(timeout_ms = 0)) {
  local({
    on.exit(mut$unlock())
    writeLines('more data', tmp)
  })
} else {
  warning('Mutex was locked. Giving up.')
}

mut$remove()
unlink(tmp)
```

`semaphore` *Increment and Decrement an Integer*

Description

A semaphore is an integer that the operating system keeps track of. Any process that knows the semaphore's identifier can increment or decrement its value, though it cannot be decremented below zero.

When the semaphore is zero, calling `$wait(timeout_ms = 0)` will return `FALSE` whereas `$wait(timeout_ms = Inf)` will block until the semaphore is incremented by another process. If multiple processes are blocked, a single call to `$post()` will only unblock one of the blocked processes.

It is possible to wait for a specific amount of time, for example, `$wait(timeout_ms = 10000)` will wait for 10 seconds. If the semaphore is incremented within those 10 seconds, the function will immediately return `TRUE`. Otherwise it will return `FALSE` at the 10 second mark.

Usage

```
semaphore(name = uid(), assert = NULL, value = 0, cleanup = FALSE, file = NULL)
```

```
## S3 method for class 'semaphore'
with(data, expr, alt_expr = NULL, timeout_ms = Inf, ...)
```

Arguments

`name`	Unique ID. Alphanumeric, starting with a letter.
`assert`	Apply an additional constraint. • `'create'` - Error if the semaphore **already exists**. • `'exists'` - Error if the semaphore **doesn't exist**. • `NULL` - No constraint; create the semaphore if it doesn't exist.
`value`	The initial value of the semaphore.
`cleanup`	Remove the semaphore when the R session exits. If `FALSE`, the semaphore will persist until `$remove()` is called or the operating system is restarted.
`file`	Use a hash of this file/directory path as the semaphore name. The file itself will not be read or modified, and does not need to exist.
`data`	A `semaphore` object.
`expr`	Expression to evaluate if a semaphore is posted.
`alt_expr`	Expression to evaluate if `timeout_ms` is reached.
`timeout_ms`	Maximum time (in milliseconds) to block the process while waiting for the operation to succeed. Use `0` or `Inf` to return immediately or only when successful, respectively.
`...`	Not used.

Value

`semaphore()` returns a `semaphore` object with the following methods:

- `$name`
  - Returns the semaphore's name (scalar character).
- `$post()`
  - Returns `TRUE` if the increment was successful, or `FALSE` on error.
- `$wait(timeout_ms = Inf)`
  - Returns `TRUE` if the decrement was successful, or `FALSE` if the timeout is reached.
- `$remove()`
  - Returns `TRUE` if the semaphore was successfully deleted from the operating system, or `FALSE` on error.

`with()` returns `eval(expr)` on success, or `eval(alt_expr)` if the timeout is reached.

See Also

Other shared objects: `msg_queue()`, `mutex()`

Examples

```
sem <- interprocess::semaphore()
print(sem)

sem$post()
sem$wait(timeout_ms = 0)
sem$wait(timeout_ms = 0)

sem$post()
with(sem, 'success', 'timed out', timeout_ms = 0)
with(sem, 'success', 'timed out', timeout_ms = 0)

sem$remove()
```

`uid` *Generate Names*

Description

To ensure broad compatibility across different operating systems, names of mutexes, semaphores, and message queues should start with a letter followed by up to 249 alphanumeric characters. These functions generate names meeting these requirements.

- `uid()`: 11-character encoding of PID and time since epoch.
- `hash()`: 11-character hash of any string (hash space = 2^64).

Usage

```
uid()

hash(str)
```

Arguments

`str`	A string (scalar character).

Details

`uid()`s encode sequential 1/100 second intervals, beginning at the current process's start time. If the number of requested UIDs exceeds the number of 1/100 seconds that the process has been alive, then the process will momentarily sleep before returning.

A `uid()` begins with an uppercase letter (`A - R`); a `hash()` begins with a lowercase letter (`a - v`).

Value

A string (scalar character) that can be used as a mutex, semaphore, or message queue name.

Examples

```
library(interprocess)

uid()

hash('192.168.1.123:8011')
```

Index