Package 'sys'

November 13, 2018

Type Package

Title Powerful and Reliable Tools for Running System Commands in R

Version 2.1

Description Drop-in replacements for the base system2() function with fine control and consistent behavior across platforms. Supports clean interruption, timeout, background tasks, and streaming STDIN / STDOUT / STDERR over binary or text connections. Arguments on Windows automatically get encoded and quoted to work on different locales. On Unix platforms the package also provides functions for evaluating expressions inside a temporary fork. Such evaluations have no side effects on the main R process, and support reliable interrupts and timeouts. This provides the basis for a sandboxing mechanism.

License MIT + file LICENSE

URL https://github.com/jeroen/sys#readme

BugReports https://github.com/jeroen/sys/issues

Encoding UTF-8

LazyData true

RoxygenNote 6.1.0

SystemRequirements libapparmor-dev (optional, debian/ubuntu only)

Suggests spelling, unix (>= 1.3), testthat

Language en-US

NeedsCompilation yes

Author Jeroen Ooms [aut, cre] (<https://orcid.org/0000-0002-4035-0289>), Gábor Csárdi [ctb]

Maintainer Jeroen Ooms <jeroen@berkeley.edu>

Repository CRAN

Date/Publication 2018-11-13 08:50:03 UTC

R topics documented:

`eval_safe` *Safe Evaluation*

Description

Evaluates an expression in a temporary fork and returns the value without any side effects on the main R session. For `eval_safe()` the expression is wrapped in additional R code to handle errors and graphics.

Usage

```
eval_safe(expr, tmp = tempfile("fork"), std_out = stdout(),
  std_err = stderr(), timeout = 0, priority = NULL, uid = NULL,
  gid = NULL, rlimits = NULL, profile = NULL, device = pdf)

eval_fork(expr, tmp = tempfile("fork"), std_out = stdout(),
  std_err = stderr(), timeout = 0)
```

Arguments

`expr`	expression to evaluate
`tmp`	the value of `tempdir()` inside the forked process
`std_out`	if and where to direct child process `STDOUT`. Must be one of `TRUE`, `FALSE`, filename, connection object or callback function. See section on *Output Streams* below for details.
`std_err`	if and where to direct child process `STDERR`. Must be one of `TRUE`, `FALSE`, filename, connection object or callback function. See section on *Output Streams* below for details.
`timeout`	maximum time in seconds to allow for call to return
`priority`	(integer) priority of the child process. High value is low priority. Non root user may only raise this value (decrease priority)
`uid`	evaluate as given user (uid or name). See `unix::setuid()`, only for root.
`gid`	evaluate as given group (gid or name). See `unix::setgid()` only for root.
`rlimits`	named vector/list with rlimit values, for example: `c(cpu = 60, fsize = 1e6)`.
`profile`	AppArmor profile, see `RAppArmor::aa_change_profile()`. Requires the `RAppArmor` package (Debian/Ubuntu only)
`device`	graphics device to use in the fork, see `dev.new()`

Details

Some programs such as `Java` are not fork-safe and cannot be called from within a forked process if they have already been loaded in the main process. On MacOS any software calling `CoreFoundation` functionality might crash within the fork. This includes `libcurl` which has been built on OSX against native SecureTransport rather than OpenSSL for https connections. The same limitations hold for e.g. `parallel::mcparallel()`.

Examples

```
#Only works on Unix
if(.Platform$OS.type == "unix"){

# works like regular eval:
eval_safe(rnorm(5))

# Exceptions get propagated
test <- function() { doesnotexit() }
tryCatch(eval_safe(test()), error = function(e){
  cat("oh no!", e$message, "\n")
})

# Honor interrupt and timeout, even inside C evaluations
try(eval_safe(svd(matrix(rnorm(1e8), 1e4)), timeout = 2))

# Capture output
outcon <- rawConnection(raw(0), "r+")
eval_safe(print(sessionInfo()), std_out = outcon)
cat(rawToChar(rawConnectionValue(outcon)))
close(outcon)
}
```

---

exec *Running System Commands*

Description

Powerful replacements for system2 with support for interruptions, background tasks and fine grained control over `STDOUT` / `STDERR` binary or text streams.

Usage

```
exec_wait(cmd, args = NULL, std_out = stdout(), std_err = stderr(),
  std_in = NULL, timeout = 0)

exec_background(cmd, args = NULL, std_out = TRUE, std_err = TRUE,
  std_in = NULL)

exec_internal(cmd, args = NULL, std_in = NULL, error = TRUE,
```

```
  timeout = 0)

exec_status(pid, wait = TRUE)
```

Arguments

cmd	the command to run. Either a full path or the name of a program on the PATH. On Windows this is automatically converted to a short path using Sys.which, unless wrapped in I().
args	character vector of arguments to pass. On Windows these automatically get quoted using windows_quote, unless the value is wrapped in I().
std_out	if and where to direct child process STDOUT. Must be one of TRUE, FALSE, filename, connection object or callback function. See section on *Output Streams* below for details.
std_err	if and where to direct child process STDERR. Must be one of TRUE, FALSE, filename, connection object or callback function. See section on *Output Streams* below for details.
std_in	file path to map std_in
timeout	maximum time in seconds
error	automatically raise an error if the exit status is non-zero.
pid	integer with a process ID
wait	block until the process completes

Details

Each value within the args vector will automatically be quoted when needed; you should not quote arguments yourself. Doing so anyway could lead to the value being quoted twice on some platforms.

The exec_wait function runs a system command and waits for the child process to exit. When the child process completes normally (either success or error) it returns with the program exit code. Otherwise (if the child process gets aborted) R raises an error. The R user can interrupt the program by sending SIGINT (press ESC or CTRL+C) in which case the child process tree is properly terminated. Output streams STDOUT and STDERR are piped back to the parent process and can be sent to a connection or callback function. See the section on *Output Streams* below for details.

The exec_background function starts the program and immediately returns the PID of the child process. This is useful for running a server daemon or background process. Because this is non-blocking, std_out and std_out can only be TRUE/FALSE or a file path. The state of the process can be checked with exec_status which returns the exit status, or NA if the process is still running. If wait = TRUE then exec_status blocks until the process completes (but can be interrupted). The child can be killed with tools::pskill.

The exec_internal function is a convenience wrapper around exec_wait which automatically captures output streams and raises an error if execution fails. Upon success it returns a list with status code, and raw vectors containing stdout and stderr data (use rawToChar for converting to text).

Value

`exec_background` returns a pid. `exec_wait` returns an exit code. `exec_internal` returns a list with exit code, stdout and stderr strings.

Output Streams

The `std_out` and `std_err` parameters are used to control how output streams of the child are processed. Possible values for both foreground and background processes are:

- `TRUE`: print child output in R console
- `FALSE`: suppress output stream
- *string*: name or path of file to redirect output

In addition the `exec_wait` function also supports the following `std_out` and `std_err` types:

- *connection* a writable R connection object such as stdout or stderr
- *function*: callback function with one argument accepting a raw vector (use rawToChar to convert to text).

When using `exec_background` with `std_out = TRUE` or `std_err = TRUE` on Windows, separate threads are used to print output. This works in RStudio and RTerm but not in RGui because the latter has a custom I/O mechanism. Directing output to a file is usually the safest option.

See Also

Base system2 and pipe provide other methods for running a system command with output.

Other sys: `exec_r`

Examples

```
# Run a command (interrupt with CTRL+C)
status <- exec_wait("date")

# Capture std/out
out <- exec_internal("date")
print(out$status)
cat(rawToChar(out$stdout))

if(nchar(Sys.which("ping"))){

# Run a background process (daemon)
pid <- exec_background("ping", "localhost")

# Kill it after a while
Sys.sleep(2)
tools::pskill(pid)

# Cleans up the zombie proc
exec_status(pid)
rm(pid)
}
```

exec_r *Execute R from R*

Description

Convenience wrappers for exec_wait and exec_internal that shell out to R itself: `R.home("bin/R")`.

Usage

```
r_wait(args = "--vanilla", std_out = stdout(), std_err = stderr(),
  std_in = NULL)

r_internal(args = "--vanilla", std_in = NULL, error = TRUE)

r_background(args = "--vanilla", std_out = TRUE, std_err = TRUE,
  std_in = NULL)
```

Arguments

args	command line arguments for R
std_out	if and where to direct child process STDOUT. Must be one of TRUE, FALSE, filename, connection object or callback function. See section on *Output Streams* below for details.
std_err	if and where to direct child process STDERR. Must be one of TRUE, FALSE, filename, connection object or callback function. See section on *Output Streams* below for details.
std_in	a file to send to stdin, usually an R script (see examples).
error	automatically raise an error if the exit status is non-zero.

Details

This is a simple but robust way to invoke R commands in a separate process. Use the callr package if you need more sophisticated control over (multiple) R process jobs.

See Also

Other sys: exec

Examples

```
# Hello world
r_wait("--version")

# Run some code
r_wait(c('--vanilla', '-q', '-e', 'sessionInfo()'))

# Run a script via stdin
```

```
tmp <- tempfile()
writeLines(c("x <- rnorm(100)", "mean(x)"), con = tmp)
r_wait(std_in = tmp)
```

`quote` *Quote arguments on Windows*

Description

Quotes and escapes shell arguments when needed so that they get properly parsed by most Windows programs. Algorithm is ported to R from libuv.

Usage

```
windows_quote(args)
```

Arguments

`args`	character vector with arguments

`sys_config` *Package config*

Description

Shows which features are enabled in the package configuration.

Usage

```
sys_config()

aa_config()
```

Examples

```
sys_config()
```

Index