

Package ‘sys’

June 24, 2017

Type Package

Title Portable System Utilities

Version 1.4

Description Powerful replacements for base system2 with consistent behavior across platforms. Supports interruption, background tasks, and full control over 'STDOUT' / 'STDERR' binary or text streams. On Unix systems the package also has 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 6.0.1

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

Suggests unix (>= 1.3), testthat

NeedsCompilation yes

Author Jeroen Ooms [aut, cre],
Gábor Csárdi [ctb]

Maintainer Jeroen Ooms <jeroen@berkeley.edu>

Repository CRAN

Date/Publication 2017-06-24 21:55:26 UTC

R topics documented:

eval_safe	2
exec	3
sys_config	5

Index	6
--------------	----------

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

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

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::mcpipeline()`.

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)))
}
```

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())
```

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

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

```
exec_status(pid, wait = TRUE)
```

Arguments

cmd	the command to run. Either a full path or the name of a program which exists in the PATH.
args	character vector of arguments to pass
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 <i>Output Streams</i> below for details.

<code>std_err</code>	if and where to direct child process STDERR. Must be one of TRUE, FALSE, file-name, connection object or callback function. See section on <i>Output Streams</i> below for details.
<code>error</code>	automatically raise an error if the exit status is non-zero.
<code>pid</code>	integer with a process ID
<code>wait</code>	block until the process completes

Details

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_err` 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 writeable 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.

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)
}
```

sys_config

Package config

Description

Shows which features are enabled in the package configuration.

Usage

```
sys_config()
```

```
aa_config()
```

Examples

```
sys_config()
```

Index

`aa_config (sys_config)`, 5

`connection`, 4

`dev.new()`, 2

`eval_fork (eval_safe)`, 2

`eval_safe`, 2

`eval_safe()`, 2

`exec`, 3

`exec_background (exec)`, 3

`exec_internal (exec)`, 3

`exec_status (exec)`, 3

`exec_wait (exec)`, 3

`pipe`, 5

`rawToChar`, 4

`stderr`, 4

`stdout`, 4

`sys (exec)`, 3

`sys_config`, 5

`system2`, 3, 5

`tempdir()`, 2

`tools::pskill`, 4

`unix::setgid()`, 2

`unix::setuid()`, 2