

# Package ‘ps’

November 6, 2018

**Version** 1.2.1

**Title** List, Query, Manipulate System Processes

**Description** List, query and manipulate all system processes, on 'Windows', 'Linux' and 'macOS'.

**License** BSD\_3\_clause + file LICENSE

**URL** <https://github.com/r-lib/ps#readme>

**BugReports** <https://github.com/r-lib/ps/issues>

**Encoding** UTF-8

**Depends** R (>= 3.1)

**Imports** utils

**Suggests** callr,  
covr,  
curl,  
pingr,  
processx (>= 3.1.0),  
R6,  
rlang,  
testthat,  
tibble

**RoxygenNote** 6.1.0

**Roxygen** list(markdown = TRUE)

**Biarch** true

## R topics documented:

CleanupReporter . . . . .	2
ps . . . . .	4
ps_boot_time . . . . .	4
ps_children . . . . .	5
ps_cmdline . . . . .	5
ps_connections . . . . .	6
ps_cpu_times . . . . .	7
ps_create_time . . . . .	8
ps_cwd . . . . .	9
ps_envIRON . . . . .	9

ps_exe . . . . .	10
ps_handle . . . . .	11
ps_interrupt . . . . .	12
ps_is_running . . . . .	12
ps_kill . . . . .	13
ps_mark_tree . . . . .	14
ps_memory_info . . . . .	15
ps_name . . . . .	16
ps_num_fds . . . . .	17
ps_num_threads . . . . .	18
ps_open_files . . . . .	18
ps_os_type . . . . .	19
ps_pid . . . . .	20
ps_pids . . . . .	20
ps_ppid . . . . .	21
ps_resume . . . . .	22
ps_send_signal . . . . .	22
ps_status . . . . .	23
ps_suspend . . . . .	24
ps_terminal . . . . .	25
ps_terminate . . . . .	25
ps_uids . . . . .	26
ps_username . . . . .	27
ps_users . . . . .	28
signals . . . . .	28

<b>Index</b>	<b>29</b>
--------------	-----------

---

CleanupReporter	<i>testthat reporter that checks if child processes are cleaned up in tests</i>
-----------------	---

---

## Description

CleanupReporter takes an existing testthat Reporter object, and wraps it, so it checks for leftover child processes, at the specified place, see the `proc_unit` argument below.

## Usage

```
CleanupReporter(reporter = testthat::ProgressReporter)
```

## Arguments

`reporter` A testthat reporter to wrap into a new CleanupReporter class.

## Details

Child processes can be reported via a failed expectation, cleaned up silently, or cleaned up and reported (the default).

The constructor of the CleanupReporter class has options:

- `file`: the output file, if any, this is passed to reporter.
- `proc_unit`: when to perform the child process check and cleanup. Possible values:

- "test": at the end of each `testthat::test_that()` block (the default),
- "testsuite": at the end of the test suite.
- `proc_cleanup`: Logical scalar, whether to kill the leftover processes, TRUE by default.
- `proc_fail`: Whether to create an expectation, that fails if there are any processes alive, TRUE by default.
- `proc_timeout`: How long to wait for the processes to quit. This is sometimes needed, because even if some kill signals were sent to child processes, it might take a short time for these to take effect. It defaults to one second.
- `rconn_unit`: When to perform the R connection cleanup. Possible values are "test" and "testsuite", like for `proc_unit`.
- `rconn_cleanup`: Logical scalar, whether to clean up leftover R connections. TRUE by default.
- `rconn_fail`: Whether to fail for leftover R connections. TRUE by default.
- `file_unit`: When to check for open files. Possible values are "test" and "testsuite", like for `proc_unit`.
- `file_fail`: Whether to fail for leftover open files. TRUE by default.
- `conn_unit`: When to check for open network connections. Possible values are "test" and "testsuite", like for `proc_unit`.
- `conn_fail`: Whether to fail for leftover network connections. TRUE by default.

### Value

New reporter class that behaves exactly like `reporter`, but it checks for, and optionally cleans up child processes, at the specified granularity.

### Examples

This is how to use this reporter in `testthat.R`:

```
library(testthat)
library(mypackage)

if (ps::ps_is_supported()) {
  reporter <- ps::CleanupReporter(testthat::ProgressReporter)$new(
    proc_unit = "test", proc_cleanup = TRUE)
} else {
  ## ps does not support this platform
  reporter <- "progress"
}

test_check("mypackage", reporter = reporter)
```

### Note

Some IDEs, like RStudio, start child processes frequently, and sometimes crash when these are killed, only use this reporter in a terminal session. In particular, you can always use it in the idiomatic `testthat.R` file, that calls `test_check()` during R CMD check.

---

ps *Process table*

---

### Description

Process table

### Usage

```
ps(user = NULL, after = NULL)
```

### Arguments

user	Username, to filter the results to matching processes.
after	Start time (POSIXt), to filter the results to processes that started after this.

### Value

Data frame (tibble), see columns below.

Columns:

- pid: Process ID.
- ppid: Process ID of parent process.
- name: Process name.
- username: Name of the user (real uid on POSIX).
- status: I.e. *running*, *sleeping*, etc.
- user: User CPU time.
- system: System CPU time.
- rss: Resident set size, the amount of memory the process currently uses. Does not include memory that is swapped out. It does include shared libraries.
- vms: Virtual memory size. All memory the process has access to.
- created: Time stamp when the process was created.
- ps\_handle: ps\_handle objects, in a list column.

---

ps\_boot\_time *Boot time of the system*

---

### Description

Boot time of the system

### Usage

```
ps_boot_time()
```

### Value

A POSIXct object.

---

ps_children	<i>List of child processes (process objects) of the process. Note that this typically requires enumerating all processes on the system, so it is a costly operation.</i>
-------------	--

---

**Description**

List of child processes (process objects) of the process. Note that this typically requires enumerating all processes on the system, so it is a costly operation.

**Usage**

```
ps_children(p, recursive = FALSE)
```

**Arguments**

p	Process handle.
recursive	Whether to include the children of the children, etc.

**Value**

List of ps\_handle objects.

**Examples**

```
p <- ps_parent(ps_handle())
ps_children(p)
```

**See Also**

Other process handle functions: [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_envIRON](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps_cmdline	<i>Command line of the process</i>
------------	------------------------------------

---

**Description**

Command line of the process, i.e. the executable and the command line arguments, in a character vector. On Unix the program might change its command line, and some programs actually do it.

**Usage**

```
ps_cmdline(p)
```

**Arguments**

p	Process handle.
---	-----------------

**Details**

For a zombie process it throws a `zombie_process` error.

**Value**

Character vector.

**Examples**

```
p <- ps_handle()
p
ps_name(p)
ps_exe(p)
ps_cmdline(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_connections

*List network connections of a process*

---

**Description**

For a zombie process it throws a `zombie_process` error.

**Usage**

```
ps_connections(p)
```

**Arguments**

`p`                      Process handle.

**Value**

Data frame, or tibble if the *tibble* package is available, with columns:

- `fd`: integer file descriptor on POSIX systems, NA on Windows.
- `family`: Address family, string, typically `AF_UNIX`, `AF_INET` or `AF_INET6`.
- `type`: Socket type, string, typically `SOCK_STREAM` (TCP) or `SOCK_DGRAM` (UDP).
- `laddr`: Local address, string, NA for UNIX sockets.
- `lport`: Local port, integer, NA for UNIX sockets.
- `raddr`: Remote address, string, NA for UNIX sockets. This is always NA for `AF_INET` sockets on Linux.
- `rport`: Remote port, integer, NA for UNIX sockets.
- `state`: Socket state, e.g. `CONN_ESTABLISHED`, etc. It is NA for UNIX sockets.

### Examples

```
p <- ps_handle()
ps_connections(p)
sc <- socketConnection("httpbin.org", port = 80)
ps_connections(p)
close(sc)
ps_connections(p)
```

### See Also

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_envIRON](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_cpu\_times

*CPU times of the process*

---

### Description

All times are measured in seconds:

- user: Amount of time that this process has been scheduled in user mode.
- system: Amount of time that this process has been scheduled in kernel mode
- children\_user: On Linux, amount of time that this process's waited-for children have been scheduled in user mode.
- children\_system: On Linux, Amount of time that this process's waited-for children have been scheduled in kernel mode.

### Usage

```
ps_cpu_times(p)
```

### Arguments

p                    Process handle.

### Details

Throws a `zombie_process()` error for zombie processes.

### Value

Named real vector or length four: user, system, children\_user, children\_system. The last two are NA on non-Linux systems.

### Examples

```
p <- ps_handle()
p
ps_cpu_times(p)
proc.time()
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_envIRON](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

<code>ps_create_time</code>	<i>Start time of a process</i>
-----------------------------	--------------------------------

---

**Description**

The pid and the start time pair serves as the identifier of the process, as process ids might be reused, but the chance of starting two processes with identical ids within the resolution of the timer is minimal.

**Usage**

```
ps_create_time(p)
```

**Arguments**

<code>p</code>	Process handle.
----------------	-----------------

**Details**

This function works even if the process has already finished.

**Value**

POSIXct object, start time, in GMT.

**Examples**

```
p <- ps_handle()
p
ps_create_time(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_cwd](#), [ps\\_envIRON](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)



---

ps_cwd	<i>Process current working directory as an absolute path.</i>
--------	---

---

**Description**

For a zombie process it throws a `zombie_process` error.

**Usage**

```
ps_cwd(p)
```

**Arguments**

`p`                    Process handle.

**Value**

String scalar.

**Examples**

```
p <- ps_handle()
p
ps_cwd(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_envIRON](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps_envIRON	<i>Environment variables of a process</i>
------------	---

---

**Description**

`ps_envIRON()` returns the environment variables of the process, in a named vector, similarly to the return value of `Sys.getenv()` (without arguments).

**Usage**

```
ps_envIRON(p)
```

```
ps_envIRON_raw(p)
```

**Arguments**

`p`                    Process handle.

**Details**

Note: this usually does not reflect changes made after the process started.

`ps_environ_raw()` is similar to `p$environ()` but returns the unparsed "var=value" strings. This is faster, and sometimes good enough.

These functions throw a `zombie_process` error for zombie processes.

**Value**

`ps_environ()` returns a named character vector (that has a `Dlist` class, so it is printed nicely), `ps_environ_raw()` returns a character vector.

**Examples**

```
p <- ps_handle()
p
env <- ps_environ(p)
env[["R_HOME"]]
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_exe

*Full path of the executable of a process*

---

**Description**

Path to the executable of the process. May also be an empty string or NA if it cannot be determined.

**Usage**

```
ps_exe(p)
```

**Arguments**

p                    Process handle.

**Details**

For a zombie process it throws a `zombie_process` error.

**Value**

Character scalar.

**Examples**

```
p <- ps_handle()
p
ps_name(p)
ps_exe(p)
ps_cmdline(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps_handle	<i>Create a process handle</i>
-----------	--------------------------------

---

**Description**

Create a process handle

**Usage**

```
ps_handle(pid = NULL, time = NULL)

## S3 method for class 'ps_handle'
as.character(x, ...)

## S3 method for class 'ps_handle'
format(x, ...)

## S3 method for class 'ps_handle'
print(x, ...)
```

**Arguments**

pid	Process id. Integer scalar. NULL means the current R process.
time	Start time of the process. Usually NULL and ps will query the start time.
x	Process handle.
...	Not used currently.

**Value**

ps\_handle() returns a process handle (class ps\_handle).

**Examples**

```
p <- ps_handle()
p
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps_interrupt	<i>Interrupt a process</i>
--------------	----------------------------

---

**Description**

Sends SIGINT on POSIX, and 'CTRL+C' or 'CTRL+BREAK' on Windows.

**Usage**

```
ps_interrupt(p, ctrl_c = TRUE)
```

**Arguments**

p	Process handle.
ctrl_c	On Windows, whether to send 'CTRL+C'. If FALSE, then 'CTRL+BREAK' is sent. Ignored on non-Windows platforms.

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps_is_running	<i>Checks whether a process is running</i>
---------------	--

---

**Description**

It returns FALSE if the process has already finished.

**Usage**

```
ps_is_running(p)
```

**Arguments**

p	Process handle.
---	-----------------

**Details**

It uses the start time of the process to work around pid reuse. I.e.

**Value**

Logical scalar.

**Examples**

```
p <- ps_handle()
p
ps_is_running(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_envIRON](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_kill

*Kill a process*

---

**Description**

Kill the current process with SIGKILL preemptively checking whether PID has been reused. On Windows it uses TerminateProcess().

**Usage**

```
ps_kill(p)
```

**Arguments**

p                    Process handle.

**Examples**

```
px <- processx::process$new("sleep", "10")
p <- ps_handle(px$get_pid())
p
ps_kill(p)
p
ps_is_running(p)
px$get_exit_status()
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_envIRON](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

 ps\_mark\_tree

*Mark a process and its (future) child tree*


---

### Description

ps\_mark\_tree() generates a random environment variable name and sets it in the current R process. This environment variable will be (by default) inherited by all child (and grandchild, etc.) processes, and will help finding these processes, even if and when they are (no longer) related to the current R process. (I.e. they are not connected in the process tree.)

### Usage

```
ps_mark_tree()
```

```
with_process_cleanup(expr)
```

```
ps_find_tree(marker)
```

```
ps_kill_tree(marker, sig = signals())$SIGKILL)
```

### Arguments

expr	R expression to evaluate in the new context.
marker	String scalar, the name of the environment variable to use to find the marked processes.
sig	The signal to send to the marked processes on Unix. On Windows this argument is ignored currently.

### Details

ps\_find\_tree() finds the processes that set the supplied environment variable and returns them in a list.

ps\_kill\_tree() finds the processes that set the supplied environment variable, and kills them (or sends them the specified signal on Unix).

with\_process\_cleanup() evaluates an R expression, and cleans up all external processes that were started by the R process while evaluating the expression. This includes child processes of child processes, etc., recursively. It returns a list with entries: result is the result of the expression, visible is TRUE if the expression should be printed to the screen, and process\_cleanup is a named integer vector of the cleaned pids, names are the process names.

If expr throws an error, then so does with\_process\_cleanup(), the same error. Nevertheless processes are still cleaned up.

### Value

ps\_mark\_tree() returns the name of the environment variable, which can be used as the marker in ps\_kill\_tree().

ps\_find\_tree() returns a list of ps\_handle objects.

ps\_kill\_tree() returns the pids of the killed processes, in a named integer vector. The names are the file names of the executables, when available.

with\_process\_cleanup() returns the value of the evaluated expression.

---

ps_memory_info	<i>Memory usage information</i>
----------------	---------------------------------

---

## Description

A list with information about memory usage. Portable fields:

- `rss`: "Resident Set Size", this is the non-swapped physical memory a process has used. On UNIX it matches "top"'s 'RES' column (see doc). On Windows this is an alias for `wset` field and it matches "Memory" column of `taskmgr.exe`.
- `vmem`: "Virtual Memory Size", this is the total amount of virtual memory used by the process. On UNIX it matches "top"'s 'VIRT' column (see doc). On Windows this is an alias for the `pagefile` field and it matches the "Working set (memory)" column of `taskmgr.exe`.

## Usage

```
ps_memory_info(p)
```

## Arguments

<code>p</code>	Process handle.
----------------	-----------------

## Details

Non-portable fields:

- `shared`: (Linux) memory that could be potentially shared with other processes. This matches "top"'s 'SHR' column (see doc).
- `text`: (Linux): aka 'TRS' (text resident set) the amount of memory devoted to executable code. This matches "top"'s 'CODE' column (see doc).
- `data`: (Linux): aka 'DRS' (data resident set) the amount of physical memory devoted to other than executable code. It matches "top"'s 'DATA' column (see doc).
- `lib`: (Linux): the memory used by shared libraries.
- `dirty`: (Linux): the number of dirty pages.
- `pfaults`: (macOS): number of page faults.
- `pageins`: (macOS): number of actual pageins.

For an explanation of Windows fields see the [PROCESS\\_MEMORY\\_COUNTERS\\_EX](#) structure.

Throws a `zombie_process()` error for zombie processes.

## Value

Named real vector.

## Examples

```
p <- ps_handle()
p
ps_memory_info(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps_name	<i>Process name</i>
---------	---------------------

---

**Description**

The name of the program, which is typically the name of the executable.

**Usage**

```
ps_name(p)
```

**Arguments**

p                   Process handle.

**Details**

On on Unix this can change, e.g. via an `exec*()` system call.

`ps_name()` works on zombie processes.

**Value**

Character scalar.

**Examples**

```
p <- ps_handle()
p
ps_name(p)
ps_exe(p)
ps_cmdline(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)



---

ps_num_fds	<i>Number of open file descriptors</i>
------------	--

---

## Description

Note that in some IDEs, e.g. RStudio or R.app on macOS, the IDE itself opens files from other threads, in addition to the files opened from the main R thread.

## Usage

```
ps_num_fds(p)
```

## Arguments

`p` Process handle.

## Details

For a zombie process it throws a `zombie_process` error.

## Value

Integer scalar.

## Examples

```
p <- ps_handle()
ps_num_fds(p)
f <- file(tmp <- tempfile(), "w")
ps_num_fds(p)
close(f)
unlink(tmp)
ps_num_fds(p)
```

## See Also

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps_num_threads	<i>Number of threads</i>
----------------	--------------------------

---

**Description**

Throws a `zombie_process()` error for zombie processes.

**Usage**

```
ps_num_threads(p)
```

**Arguments**

`p` Process handle.

**Value**

Integer scalar.

**Examples**

```
p <- ps_handle()
p
ps_num_threads(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps_open_files	<i>Open files of a process</i>
---------------	--------------------------------

---

**Description**

Note that in some IDEs, e.g. RStudio or R.app on macOS, the IDE itself opens files from other threads, in addition to the files opened from the main R thread.

**Usage**

```
ps_open_files(p)
```

**Arguments**

`p` Process handle.

**Details**

For a zombie process it throws a `zombie_process` error.

**Value**

Data frame, or tibble if the *tibble* package is available, with columns: fd and path. fd is numeric file descriptor on POSIX systems, NA on Windows. path is an absolute path to the file.

**Examples**

```
p <- ps_handle()
ps_open_files(p)
f <- file(tmp <- tempfile(), "w")
ps_open_files(p)
close(f)
unlink(tmp)
ps_open_files(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_os\_type

*Query the type of the OS*

---

**Description**

Query the type of the OS

**Usage**

```
ps_os_type()

ps_is_supported()
```

**Value**

ps\_os\_type returns a named logical vector. The rest of the functions return a logical scalar.

ps\_is\_supported() returns TRUE if ps supports the current platform.

**Examples**

```
ps_os_type()
ps_is_supported()
```

ps\_pid

*Pid of a process handle*

---

**Description**

This function works even if the process has already finished.

**Usage**

```
ps_pid(p)
```

**Arguments**

p                    Process handle.

**Value**

Process id.

**Examples**

```
p <- ps_handle()
p
ps_pid(p)
ps_pid(p) == Sys.getpid()
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_pids*Ids of all processes on the system*

---

**Description**

Ids of all processes on the system

**Usage**

```
ps_pids()
```

**Value**

Integer vector of process ids.

---

ps_ppid	<i>Parent pid or parent process of a process</i>
---------	--

---

### Description

ps\_ppid() returns the parent pid, ps\_parent() returns a ps\_handle of the parent.

### Usage

```
ps_ppid(p)
```

```
ps_parent(p)
```

### Arguments

p                    Process handle.

### Details

On POSIX systems, if the parent process terminates, another process (typically the pid 1 process) is marked as parent. ps\_ppid() and ps\_parent() will return this process then.

Both ps\_ppid() and ps\_parent() work for zombie processes.

### Value

ps\_ppid() returns an integer scalar, the pid of the parent of p. ps\_parent() returns a ps\_handle.

### Examples

```
p <- ps_handle()
p
ps_ppid(p)
ps_parent(p)
```

### See Also

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_resume                      *Resume (continue) a stopped process*

---

### Description

Resume process execution with SIGCONT preemptively checking whether PID has been reused. On Windows this has the effect of resuming all process threads.

### Usage

```
ps_resume(p)
```

### Arguments

p                      Process handle.

### Examples

```
px <- processx::process$new("sleep", "10")
p <- ps_handle(px$get_pid())
p
ps_suspend(p)
ps_status(p)
ps_resume(p)
ps_status(p)
ps_kill(p)
```

### See Also

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_send\_signal                      *Send signal to a process*

---

### Description

Send a signal to the process. Not implemented on Windows. See [signals\(\)](#) for the list of signals on the current platform.

### Usage

```
ps_send_signal(p, sig)
```

### Arguments

p                      Process handle.  
sig                      Signal number, see [signals\(\)](#).

**Details**

It checks if the process is still running, before sending the signal, to avoid signalling the wrong process, because of pid reuse.

**Examples**

```
px <- processx::process$new("sleep", "10")
p <- ps_handle(px$get_pid())
p
ps_send_signal(p, signals())$SIGINT)
p
ps_is_running(p)
px$get_exit_status()
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_status

*Current process status*

---

**Description**

One of the following:

- "idle": Process being created by fork, macOS only.
- "running": Currently runnable on macOS and Windows. Actually running on Linux.
- "sleeping" Sleeping on a wait or poll.
- "disk\_sleep" Uninterruptible sleep, waiting for an I/O operation (Linux only).
- "stopped" Stopped, either by a job control signal or because it is being traced.
- "tracing\_stop" Stopped for tracing (Linux only).
- "zombie" Zombie. Finished, but parent has not read out the exit status yet.
- "dead" Should never be seen (Linux).
- "wake\_kill" Received fatal signal (Linux only).
- "waking" Paging (Linux only, not valid since the 2.6.xx kernel).

**Usage**

```
ps_status(p)
```

**Arguments**

p                      Process handle.

**Details**

Works for zombie processes.

**Value**

Character scalar.

**Examples**

```
p <- ps_handle()
p
ps_status(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_suspend

*Suspend (stop) the process*

---

**Description**

Suspend process execution with SIGSTOP preemptively checking whether PID has been reused. On Windows this has the effect of suspending all process threads.

**Usage**

```
ps_suspend(p)
```

**Arguments**

p                    Process handle.

**Examples**

```
px <- processx::process$new("sleep", "10")
p <- ps_handle(px$get_pid())
p
ps_suspend(p)
ps_status(p)
ps_resume(p)
ps_status(p)
ps_kill(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_terminal](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)



---

ps_terminal	<i>Terminal device of the process</i>
-------------	---------------------------------------

---

**Description**

Returns the terminal of the process. Not implemented on Windows, always returns NA\_character\_. On Unix it returns NA\_character\_ if the process has no terminal.

**Usage**

```
ps_terminal(p)
```

**Arguments**

p                    Process handle.

**Details**

Works for zombie processes.

**Value**

Character scalar.

**Examples**

```
p <- ps_handle()
p
ps_terminal(p)
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminate](#), [ps\\_uids](#), [ps\\_username](#)

---

ps_terminate	<i>Terminate a Unix process</i>
--------------	---------------------------------

---

**Description**

Send a SIGTERM signal to the process. Not implemented on Windows.

**Usage**

```
ps_terminate(p)
```

**Arguments**

p                    Process handle.

**Details**

Checks if the process is still running, to work around pid reuse.

**Examples**

```
px <- processx::process$new("sleep", "10")
p <- ps_handle(px$get_pid())
p
ps_terminate(p)
p
ps_is_running(p)
px$get_exit_status()
```

**See Also**

Other process handle functions: [ps\\_children](#), [ps\\_cmdline](#), [ps\\_connections](#), [ps\\_cpu\\_times](#), [ps\\_create\\_time](#), [ps\\_cwd](#), [ps\\_environ](#), [ps\\_exe](#), [ps\\_handle](#), [ps\\_interrupt](#), [ps\\_is\\_running](#), [ps\\_kill](#), [ps\\_memory\\_info](#), [ps\\_name](#), [ps\\_num\\_fds](#), [ps\\_num\\_threads](#), [ps\\_open\\_files](#), [ps\\_pid](#), [ps\\_ppid](#), [ps\\_resume](#), [ps\\_send\\_signal](#), [ps\\_status](#), [ps\\_suspend](#), [ps\\_terminal](#), [ps\\_uids](#), [ps\\_username](#)

---

ps\_uids

*User ids and group ids of the process*


---

**Description**

User ids and group ids of the process. Both return integer vectors with names: real, effective and saved.

**Usage**

```
ps_uids(p)
```

```
ps_gids(p)
```

**Arguments**

p                    Process handle.

**Details**

Both work for zombie processes.

They are not implemented on Windows, they throw a not\_implemented error.

**Value**

Named integer vector of length 3, with names: real, effective and saved.

**Examples**

```
p <- ps_handle()
p
ps_uids(p)
ps_gids(p)
```

**See Also**

`ps_username()` returns a user *name* and works on all platforms.

Other process handle functions: `ps_children`, `ps_cmdline`, `ps_connections`, `ps_cpu_times`, `ps_create_time`, `ps_cwd`, `ps_environ`, `ps_exe`, `ps_handle`, `ps_interrupt`, `ps_is_running`, `ps_kill`, `ps_memory_info`, `ps_name`, `ps_num_fds`, `ps_num_threads`, `ps_open_files`, `ps_pid`, `ps_ppid`, `ps_resume`, `ps_send_signal`, `ps_status`, `ps_suspend`, `ps_terminal`, `ps_terminate`, `ps_username`

Other process handle functions: `ps_children`, `ps_cmdline`, `ps_connections`, `ps_cpu_times`, `ps_create_time`, `ps_cwd`, `ps_environ`, `ps_exe`, `ps_handle`, `ps_interrupt`, `ps_is_running`, `ps_kill`, `ps_memory_info`, `ps_name`, `ps_num_fds`, `ps_num_threads`, `ps_open_files`, `ps_pid`, `ps_ppid`, `ps_resume`, `ps_send_signal`, `ps_status`, `ps_suspend`, `ps_terminal`, `ps_terminate`, `ps_username`

---

ps_username	<i>Owner of the process</i>
-------------	-----------------------------

---

**Description**

The name of the user that owns the process. On Unix it is calculated from the real user id.

**Usage**

```
ps_username(p)
```

**Arguments**

p                    Process handle.

**Details**

On Unix, a numeric uid id returned if the uid is not in the user database, thus a username cannot be determined.

Works for zombie processes.

**Value**

String scalar.

**Examples**

```
p <- ps_handle()
p
ps_username(p)
```

**See Also**

Other process handle functions: `ps_children`, `ps_cmdline`, `ps_connections`, `ps_cpu_times`, `ps_create_time`, `ps_cwd`, `ps_environ`, `ps_exe`, `ps_handle`, `ps_interrupt`, `ps_is_running`, `ps_kill`, `ps_memory_info`, `ps_name`, `ps_num_fds`, `ps_num_threads`, `ps_open_files`, `ps_pid`, `ps_ppid`, `ps_resume`, `ps_send_signal`, `ps_status`, `ps_suspend`, `ps_terminal`, `ps_terminate`, `ps_uids`

---

ps_users	<i>List users connected to the system</i>
----------	---

---

**Description**

List users connected to the system

**Usage**

```
ps_users()
```

**Value**

A data frame (tibble) with columns `username`, `tty`, `hostname`, `start_time`, `pid`. `tty` and `pid` are NA on Windows. `pid` is the process id of the login process. For local users the `hostname` column is the empty string.

---

signals	<i>List of all supported signals</i>
---------	--------------------------------------

---

**Description**

Only the signals supported by the current platform are included.

**Usage**

```
signals()
```

**Value**

List of integers, named by signal names.

# Index

`as.character.ps_handle` (`ps_handle`), 11

`CleanupReporter`, 2

`format.ps_handle` (`ps_handle`), 11

`print.ps_handle` (`ps_handle`), 11

`ps`, 4

`ps_boot_time`, 4

`ps_children`, 5, 6–13, 16–27

`ps_cmdline`, 5, 5, 7–13, 16–27

`ps_connections`, 5, 6, 6, 8–13, 16–27

`ps_cpu_times`, 5–7, 7, 8–13, 16–27

`ps_create_time`, 5–8, 8, 9–13, 16–27

`ps_cwd`, 5–8, 9, 10–13, 16–27

`ps_envIRON`, 5–9, 9, 11–13, 16–27

`ps_envIRON_raw` (`ps_envIRON`), 9

`ps_exe`, 5–10, 10, 12, 13, 16–27

`ps_find_tree` (`ps_mark_tree`), 14

`ps_gids` (`ps_uids`), 26

`ps_handle`, 5–11, 11, 12, 13, 16–27

`ps_INTERRUPT`, 5–12, 12, 13, 16–27

`ps_is_running`, 5–12, 12, 13, 16–27

`ps_is_supported` (`ps_os_type`), 19

`ps_kill`, 5–13, 13, 16–27

`ps_kill_tree` (`ps_mark_tree`), 14

`ps_mark_tree`, 14

`ps_memory_info`, 5–13, 15, 16–27

`ps_name`, 5–13, 16, 16, 17–27

`ps_num_fds`, 5–13, 16, 17, 18–27

`ps_num_threads`, 5–13, 16, 17, 18, 19–27

`ps_open_files`, 5–13, 16–18, 18, 20–27

`ps_os_type`, 19

`ps_parent` (`ps_ppid`), 21

`ps_pid`, 5–13, 16–19, 20, 21–27

`ps_pids`, 20

`ps_ppid`, 5–13, 16–20, 21, 22–27

`ps_resume`, 5–13, 16–21, 22, 23–27

`ps_send_signal`, 5–13, 16–22, 22, 24–27

`ps_status`, 5–13, 16–23, 23, 24–27

`ps_suspend`, 5–13, 16–24, 24, 25–27

`ps_terminal`, 5–13, 16–24, 25, 26, 27

`ps_terminate`, 5–13, 16–25, 25, 27

`ps_uids`, 5–13, 16–26, 26, 27

`ps_username`, 5–13, 16–27, 27

`ps_username()`, 27

`ps_users`, 28

`signals`, 28

`signals()`, 22

`testthat::test_that()`, 3

`with_process_cleanup` (`ps_mark_tree`), 14