Package: unix (via r-universe)

October 28, 2026

Title POSIX System Utilities

Version 1.6.0

Description Bindings to system utilities found in most Unix systems such as POSIX functions which are not part of the Standard C Library.

License MIT + file LICENSE

URL https://jeroen.r-universe.dev/unix

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

OS_type unix

SystemRequirements POSIX.1-2001, AppArmor (optional)

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

Suggests testthat

Language en-US

Encoding UTF-8

Config/pak/sysreqs libapparmor-dev

Repository https://jeroen.r-universe.dev

Date/Publication 2026-04-23 08:55:11 UTC

RemoteUrl https://github.com/jeroen/unix

RemoteRef HEAD

RemoteSha fa5bbd2f3da27cafaeb07238339569ab6dbdf6f5

Contents

chroot 2
eval_safe 2
getuid 4
rlimit 6
sys_config 8
userinfo 8

Index **9**

`chroot` *Change Root Dir*

Description

Changes the root directory of the calling process to that specified in path. This directory will be used for pathnames beginning with `/`. **Only a privileged process (i.e. sudo) may call** `chroot()`.

Usage

```
chroot(path = getwd())
```

Arguments

`path`	directory of the new root

Details

This call changes an ingredient in the pathname resolution process and does nothing else. In particular, it is not intended to be used for any kind of security purpose, neither to fully sandbox a process nor to restrict filesystem system calls.

References

CHROOT(2)

`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. Non root user may only raise this value (decrease priority)
timeout	maximum time in seconds to allow for call to return
priority	(integer) priority of the child process. High value is low priority.
uid	evaluate as given user (uid or name). See `setuid()`, only for root.
gid	evaluate as given group (gid or name). See `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

```
# 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)
```

---

getuid *Process Info*

Description

Get or set attributes of the current process.

Usage

```
getuid()

getgid()

geteuid()

getegid()

getpid()

getppid()

getpgid()

getpriority()

setuid(uid)

seteuid(uid)

setgid(gid)

setegid(gid)
```

```
setpgid(pgid = 0)

setpriority(prio)

kill(pid, signal = SIGTERM)
```

Arguments

`uid`	User ID from `/etc/passwd`.
`gid`	Group ID from `/etc/group`.
`pgid`	Process Group ID. Default `0` sets pgid to the current pid.
`prio`	Priority level
`pid`	process ID (integer)
`signal`	a signal number (integer), defaults to tools::SIGTERM.

Details

Acronyms stand for:

- `pid` Process ID
- `ppid` Parent-Process ID
- `pgid` Process-Group ID
- `uid` User ID
- `euid` Effective User ID
- `gid` Group ID
- `egid` Effective Group ID
- `prio` Priority level

An unprivileged (non-root) process cannot change it's `uid` and only lower process priority (higher value).

References

GETUID(2) GETPID(2) GETPGID(2) GETPRIORITY(2)

Examples

```
# Current User:
getuid()
# Current UserGroup:
getgid()
# Current UserGroup:
geteuid()
# Current UserGroup:
getegid()
# Process ID
getpid()
```

```
# parent PID:
getppid()
# Process group id:
getpgid()

# Detach process group
setpgid(0)
getpgid()
# Process priority:
getpriority()
# Decrease priority
setpriority(getpriority() + 1)
```

`rlimit` *Resource Limits*

Description

Get and set process resource limits. Each function returns the current limits, and can optionally update the limit by passing argument values. The `rlimit_all()` function is a convenience wrapper which prints all current hard and soft limits.

Usage

```
rlimit_all()

rlimit_as(cur = NULL, max = NULL)

rlimit_core(cur = NULL, max = NULL)

rlimit_cpu(cur = NULL, max = NULL)

rlimit_data(cur = NULL, max = NULL)

rlimit_fsize(cur = NULL, max = NULL)

rlimit_memlock(cur = NULL, max = NULL)

rlimit_nofile(cur = NULL, max = NULL)

rlimit_nproc(cur = NULL, max = NULL)

rlimit_stack(cur = NULL, max = NULL)
```

Arguments

`cur`	set the current (soft) limit for this resource. See details.
`max`	set the max (hard) limit for this resource. See details.

Details

Each resource has an associated soft and hard limit. The soft limit is the value that the kernel enforces for the corresponding resource. The hard limit acts as a ceiling for the soft limit: an unprivileged process may set only its soft limit to a value in the range from 0 up to the hard limit, and (irreversibly) lower its hard limit.

Definitons from the Linux manual page are as follows:

- `RLIMIT_AS` : the maximum size of the process's virtual memory (address space) in bytes.
- `RLIMIT_CORE` : the maximum size of a core file that the process may dump.
- `RLIMIT_CPU` : a limit in seconds on the amount of CPU time (**not** elapsed time) that the process may consume. When the process reaches the soft limit, it is sent a `SIGXCPU` signal.
- `RLIMIT_DATA` : the maximum size of the process's data segment (initialized data, uninitialized data, and heap).
- `RLIMIT_FSIZE` : the maximum size of files that the process may create. Attempts to extend a file beyond this limit result in delivery of a SIGXFSZ signal.
- `RLIMIT_MEMLOCK` : the maximum number of bytes of memory that may be locked into RAM.
- `RLIMIT_NOFILE` : a value one greater than the maximum file descriptor number that can be opened by this process.
- `RLIMIT_NPROC` : the maximum number of processes that can be created for the real user ID of the calling process. Upon encountering this limit, fork fails with the error EAGAIN. Not enforced for root user.
- `RLIMIT_STACK` : the maximum size of the process stack, in bytes.

Note that the support for enforcing limits very widely by system. In particular `RLIMIT_AS` has a different meaning depending on how memory allocation is managed by the operating system (and doesn't work at all on MacOS).

References

GETRLIMIT(2)

Examples

```
# Print all limits
rlimit_all()

# Get one limit
rlimit_as()

## Not run:
# Set a soft limit
lim <- rlimit_as(1e9)
print(lim)

# Reset the limit to max
rlimit_as(cur = lim$max)
```

```
# Set a hard limit (irreversible)
rlimit_as(max = 1e10)

## End(Not run)
```

`sys_config` *Package config*

Description

Shows which features are enabled in the package configuration.

Usage

```
sys_config()

aa_config()
```

Examples

```
sys_config()
```

`userinfo` *User / Group Info*

Description

Lookup a user or group info via user uid/name or group gid/name.

Usage

```
user_info(uid = getuid())

group_info(gid = getgid())
```

Arguments

`uid`	user ID (integer) or name (string)
`gid`	group ID (integer) or name (string)

References

GETPWNAM(3) GETGRNAM(3)

Examples

```
# Get info current user
user_info()
group_info()
```

Index

aa_config *(sys_config)*, 8

chroot, 2

dev.new(), *3*

eval_fork *(eval_safe)*, 2
eval_safe, 2
eval_safe(), *2*

getegid *(getuid)*, 4
geteuid *(getuid)*, 4
getgid *(getuid)*, 4
getpgid *(getuid)*, 4
getpid *(getuid)*, 4
getppid *(getuid)*, 4
getpriority *(getuid)*, 4
getuid, 4
group_info *(userinfo)*, 8

kill *(getuid)*, 4

rlimit, 6
rlimit_all *(rlimit)*, 6
rlimit_as *(rlimit)*, 6
rlimit_core *(rlimit)*, 6
rlimit_cpu *(rlimit)*, 6
rlimit_data *(rlimit)*, 6
rlimit_fsize *(rlimit)*, 6
rlimit_memlock *(rlimit)*, 6
rlimit_nofile *(rlimit)*, 6
rlimit_nproc *(rlimit)*, 6
rlimit_stack *(rlimit)*, 6

setegid *(getuid)*, 4
seteuid *(getuid)*, 4
setgid *(getuid)*, 4
setgid(), *3*
setpgid *(getuid)*, 4
setpriority *(getuid)*, 4
setuid *(getuid)*, 4
setuid(), *3*
sys_config, 8

tempdir(), *3*
tools::SIGTERM, *5*

user_info *(userinfo)*, 8
userinfo, 8