psutil-extra

Jun 12, 2021

Contents:

CHAPTER 1

Process information

All of these functions take a `proc` argument; this can either be a `psutil.Process` instance or an `int` representing a process ID.

`oneshot_proc`(*pid*)

Similar to `psutil.Process.oneshot()`, when this is used as a context manager it enables caching of values that can be retrieved by the same method for the given PID.

Parameters **`pid`** (*`int`*) – The PID of the process for which caching should be enabled.

Warning: This function differs from `psutil.Process.oneshot()` in two important ways:

- The process information cache is thread-local. This avoids concurrent modification issues.
- The caching is done by PID, not by `psutil.Process` instance, and as a result the cache will be used regardless of whether a `psutil.Process` or an integer PID is passed to the underlying function. For example:

```
with oneshot_proc(1):
    proc_getgroups(1)  # Retrieves the process information
    proc_getgroups(1)  # Uses the cached information
    proc_getgroups(psutil.Process(1))  # Also uses the cached information
```

Note: Do not use this context manager unless you plan to retrieve multiple pieces of information.

As noted in the table below, some functions, when called inside a *`oneshot_proc()`* manager, will retrieve the requested information in a way that retrieves as much additional information as possible. While this means that the cached information can then be used later for performance improvements, it is also very wasteful if the information is *not* used.

Here is a table, in the same format as psutil.Process.oneshot()'s table, that shows which methods can be grouped together for greater efficiency:

Linux	macOS	NetBSD	OpenBSD	DragonFly-BSD	FreeBSD
proc_getgroups()	proc_getgroups()	proc_getgroups()	proc_getgroups()	proc_getgroups()	proc_getgroups()[1][3]
proc_get_umask()	proc_getpgid()[1]	proc_getpgid()[1]	proc_getpgid()[1]	proc_getpgid()[1]	proc_getpgid()[1]
proc_get_sigmasks()	proc_get_sigmasks()	proc_getsid()[1]	proc_getsid()[1]	proc_getsid()[1]	proc_getsid()[1]
		proc_get_sigmasks()	proc_get_sigmasks()		proc_get_sigmasks()
				proc_getrlimit()[2]	

proc_get_umask(*proc*)

Returns the umask of the given process without changing it.

On Linux <4.7, this raises an `OSError` with `errno` set to `ENOTSUP` (Operation not supported).

Parameters **proc** (*int or psutil.Process*) – The process to get the umask for. This can be either a `psutil.Process` or an `int` representing a PID.

Returns The given process's current umask value.

Return type int

Availability: Linux (4.7+), FreeBSD

proc_getgroups(*proc*)

Returns a list of the given process's supplementary groups.

Note: Currently, on Windows Subsystem for Linux 1 (not on WSL 2), this function succeeds but always returns an empty list.

Note: On macOS, this function's behavior differs from that of os.getgroups(). Effectively, it always behaves as if the deployment target is less than 10.5.

Parameters **proc** (*int or psutil.Process*) – The process to get the supplementary group list for. This can be either a `psutil.Process` or an `int` representing a PID.

Returns A list of the given process's current supplementary groups.

Return type list[int]

Availability: Linux, macOS, FreeBSD, OpenBSD, NetBSD, DragonFlyBSD, Solaris

proc_rlimit(*proc*, *resource*, *new_limits=None*)

Get/set the given process's resource limits. This behaves identically to `psutil.Process.rlimit()`, except it is also implemented on some platforms other than Linux.

Warning: On some platforms, this function may not be able to get/set the limits atomically, or to set the soft/hard resource limits together.

[1] These functions, when called inside a `oneshot_proc()` context manager, will retrieve the requested information in a different way that collects as much extra information as possible about the process for later use.

[3] On FreeBSD, calling `proc_getgroups()` inside a `oneshot_proc()` will first attempt to retrieve the group list via a method that collects as much extra information as possible. However, this method may truncate the returned group list. In this case, `proc_getgroups()` will fall back on the normal method, which avoids truncation.

[2] On DragonFlyBSD, the first call to `proc_getrlimit()` inside a `oneshot_proc()` will retrieve all of the resource limits and cache them. Further calls to `proc_getrlimit()` will use this cache.

Aside from the potential race conditions this creates, if this function raises an error, one or both of the limits may have been changed before the error occurred.

Parameters

- **proc** (*int or psutil.Process*) – The process to get/set the resource limits for. This can be either a `psutil.Process` or an `int` representing a PID.
- **resource** (*int*) – One of the `resource.RLIMIT_*` constants representing the resource to get/set the limits for.
- **new_limits** (*None or tuple[int, int]*) – If given and not `None`, this should be a `(soft, hard)` tuple representing the new limits to set, as with `resource.setrlimit()`.

Returns A `(soft, hard)` tuple representing the previous resource limits.

Return type tuple[int, int]

Availability: Linux, FreeBSD, NetBSD

`proc_getrlimit` (*proc*, *resource*)

This behaves identically to `proc_rlimit()`, except that it only supports *getting* resource limits. This allows it to be implemented on platforms that support getting resource limits but not setting them.

Parameters

- **proc** (*int or psutil.Process*) – The process to get the resource limits for. This can be either a `psutil.Process` or an `int` representing a PID.
- **resource** (*int*) – One of the `resource.RLIMIT_*` constants representing the resource to get the current limits for.

Returns A `(soft, hard)` tuple representing the current resource limits.

Return type tuple[int, int]

Availability: Linux, FreeBSD, NetBSD, DragonFlyBSD

`proc_get_sigmasks` (*proc*, *resource*)

Get the signal masks of the given process. Returns a dataclass containing several fields:

- `pending` (not on macOS): The set of pending signals for the process.
- `blocked` (not on macOS): The set of signals that the process has blocked.
- `ignored`: The set of signals that the process has ignored.
- `caught`: The set of signals that the process has registered signal handlers for.
- `process_pending` (Linux): The set of pending signals for the entire process, not just the specified thread.

Note: Currently, on Windows Subsystem for Linux 1 (not on WSL 2), this function succeeds but always returns empty sets for all fields.

Parameters **proc** (*int or psutil.Process*) – The process to get the resource limits for. This can be either a `psutil.Process` or an `int` representing a PID.

Returns A dataclass containing the fields listed above

Availability: Linux, macOS, FreeBSD, OpenBSD, NetBSD

`proc_getpgid`(*proc*)

Get the process group ID of the given process.

On platforms where `os.getpgid()` returns EPERM for processes in other sessions, this function may still be able to get the process group ID for these processes.

Parameters **`proc`** (*`int or psutil.Process`*) – The process to get the process group ID for. This can be either a `psutil.Process` or an `int` representing a PID.

Returns The process group ID of the given process

Return type int

Availability: Linux, macOS, FreeBSD, OpenBSD, NetBSD, DragonFlyBSD, Solaris

`proc_getsid`(*proc*)

Get the session ID of the given process.

On platforms where `os.getsid()` returns EPERM for processes in other sessions, this function may still be able to get the session ID for these processes.

Parameters **`proc`** (*`int or psutil.Process`*) – The process to get the session ID for. This can be either a `psutil.Process` or an `int` representing a PID.

Returns The session ID of the given process

Return type int

Availability: Linux, macOS, FreeBSD, OpenBSD, NetBSD, DragonFlyBSD, Solaris

CHAPTER 2

Error handling

Errors raised will vary slightly between functions and across platforms (for example, `proc_rlimit()` and `proc_getrlimit()` raise `ValueError` for invalid resource values). However, here is the general rule:

`psutil.Error` subclasses (`psutil.NoSuchProcess` and `psutil.AccessDenied`) are raised if there is one that directly corresponds to the error that occured. For most other cases, an `OSError` is raised.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index