35.11. `resource` — Resource usage information

This module provides basic mechanisms for measuring and controlling system resources utilized by a program.

Symbolic constants are used to specify particular system resources and to request usage information about either the current process or its children.

An `OSError` is raised on syscall failure.

exception `resource.`**`error`**

A deprecated alias of `OSError`.

Changed in version 3.3: Following **PEP 3151**, this class was made an alias of `OSError`.

35.11.1. Resource Limits

Resources usage can be limited using the `setrlimit()` function described below. Each resource is controlled by a pair of limits: a soft limit and a hard limit. The soft limit is the current limit, and may be lowered or raised by a process over time. The soft limit can never exceed the hard limit. The hard limit can be lowered to any value greater than the soft limit, but not raised. (Only processes with the effective UID of the super-user can raise a hard limit.)

The specific resources that can be limited are system dependent. They are described in the *getrlimit(2)* man page. The resources listed below are supported when the underlying operating system supports them; resources which cannot be checked or controlled by the operating system are not defined in this module for those platforms.

`resource.`**`RLIM_INFINITY`**

Constant used to represent the limit for an unlimited resource.

`resource.`**`getrlimit`**(*resource*)

Returns a tuple `(soft, hard)` with the current soft and hard limits of *resource*. Raises `ValueError` if an invalid resource is specified, or `error` if the underlying system call fails unexpectedly.

`resource.`**`setrlimit`**(*resource*, *limits*)

Sets new limits of consumption of *resource*. The *limits* argument must be a tuple `(soft, hard)` of two integers describing the new limits. A value of `RLIM_INFINITY` can be used to request a limit that is unlimited.

Raises `ValueError` if an invalid resource is specified, if the new soft limit exceeds the hard limit, or if a process tries to raise its hard limit. Specifying a limit of `RLIM_INFINITY` when the hard or system limit for that resource is not unlimited will result in a `ValueError`. A process with the effective UID of super-user can request any valid limit value, including unlimited, but `ValueError` will still be raised if the requested limit exceeds the system imposed limit.

`setrlimit` may also raise `error` if the underlying system call fails.

`resource.`**`prlimit`**(*pid*, *resource*[, *limits*])

Combines `setrlimit()` and `getrlimit()` in one function and supports to get and set the resources limits of an arbitrary process. If *pid* is 0, then the call applies to the current process. *resource* and *limits* have the same meaning as in `setrlimit()`, except that *limits* is optional.

When *limits* is not given the function returns the *resource* limit of the process *pid*. When *limits* is given the *resource* limit of the process is set and the former resource limit is returned.

Raises `ProcessLookupError` when *pid* can't be found and `PermissionError` when the user doesn't have `CAP_SYS_RESOURCE` for the process.

Availability: Linux 2.6.36 or later with glibc 2.13 or later

New in version 3.4.

These symbols define resources whose consumption can be controlled using the `setrlimit()` and `getrlimit()` functions described below. The values of these symbols are exactly the constants used by C programs.

The Unix man page for *getrlimit(2)* lists the available resources. Note that not all systems use the same symbol or same value to denote the same resource. This module does not attempt to mask platform differences — symbols not defined for a platform will not be available from this module on that platform.

`resource.`**`RLIMIT_CORE`**

The maximum size (in bytes) of a core file that the current process can create. This may result in the creation of a partial core file if a larger core would be required to contain the entire process image.

`resource.`**`RLIMIT_CPU`**

The maximum amount of processor time (in seconds) that a process can use. If this limit is exceeded, a `SIGXCPU` signal is sent to the process. (See the `signal` module documentation for information about how to catch this signal and do something useful, e.g. flush open files to disk.)

`resource.`**`RLIMIT_FSIZE`**

The maximum size of a file which the process may create.

`resource.`**`RLIMIT_DATA`**

The maximum size (in bytes) of the process's heap.

`resource.`**`RLIMIT_STACK`**

The maximum size (in bytes) of the call stack for the current process. This only affects the stack of the main thread in a multi-threaded process.

`resource.`**`RLIMIT_RSS`**

The maximum resident set size that should be made available to the process.

`resource.`**`RLIMIT_NPROC`**

The maximum number of processes the current process may create.

`resource.`**`RLIMIT_NOFILE`**

The maximum number of open file descriptors for the current process.

`resource.`**`RLIMIT_OFILE`**

The BSD name for `RLIMIT_NOFILE`.

`resource.`**`RLIMIT_MEMLOCK`**

The maximum address space which may be locked in memory.

`resource.`**`RLIMIT_VMEM`**

The largest area of mapped memory which the process may occupy.

`resource.`**`RLIMIT_AS`**

The maximum area (in bytes) of address space which may be taken by the process.

`resource.`**`RLIMIT_MSGQUEUE`**

The number of bytes that can be allocated for POSIX message queues.

Availability: Linux 2.6.8 or later.

New in version 3.4.

`resource.`**`RLIMIT_NICE`**

The ceiling for the process’s nice level (calculated as 20 - rlim_cur).

Availability: Linux 2.6.12 or later.

New in version 3.4.

`resource.`**`RLIMIT_RTPRIO`**

The ceiling of the real-time priority.

Availability: Linux 2.6.12 or later.

New in version 3.4.

`resource.`**`RLIMIT_RTTIME`**

The time limit (in microseconds) on CPU time that a process can spend under real-time scheduling without making a blocking syscall.

Availability: Linux 2.6.25 or later.

New in version 3.4.

`resource.`**`RLIMIT_SIGPENDING`**

The number of signals which the process may queue.

Availability: Linux 2.6.8 or later.

New in version 3.4.

`resource.`**`RLIMIT_SBSIZE`**

The maximum size (in bytes) of socket buffer usage for this user. This limits the amount of network memory, and hence the amount of mbufs, that this user may hold at any time.

Availability: FreeBSD 9 or later.

New in version 3.4.

`resource.`**`RLIMIT_SWAP`**

The maximum size (in bytes) of the swap space that may be reserved or used by all of this user id’s processes. This limit is enforced only if bit 1 of the vm.overcommit sysctl is set. Please see *tuning(7)* for a complete description of this sysctl.

Availability: FreeBSD 9 or later.

New in version 3.4.

`resource.`**`RLIMIT_NPTS`**

The maximum number of pseudo-terminals created by this user id.

Availability: FreeBSD 9 or later.

New in version 3.4.

35.11.2. Resource Usage

These functions are used to retrieve resource usage information:

`resource.`**`getrusage`**(*who*)

This function returns an object that describes the resources consumed by either the current process or its children, as specified by the *who* parameter. The *who* parameter should be specified using one of the `RUSAGE_*` constants described below.

The fields of the return value each describe how a particular system resource has been used, e.g. amount of time spent running is user mode or number of times the process was swapped out of main memory. Some values are dependent on the clock tick internal, e.g. the amount of memory the process is using.

For backward compatibility, the return value is also accessible as a tuple of 16 elements.

The fields `ru_utime` and `ru_stime` of the return value are floating point values representing the amount of time spent executing in user mode and the amount of time spent executing in system mode, respectively. The remaining values are integers. Consult the *getrusage(2)* man page for detailed information about these values. A brief summary is presented here:

Index	Field	Resource
`0`	`ru_utime`	time in user mode (float)
`1`	`ru_stime`	time in system mode (float)
`2`	`ru_maxrss`	maximum resident set size
`3`	`ru_ixrss`	shared memory size
`4`	`ru_idrss`	unshared memory size
`5`	`ru_isrss`	unshared stack size
`6`	`ru_minflt`	page faults not requiring I/O
`7`	`ru_majflt`	page faults requiring I/O

Index	Field	Resource
8	`ru_nswap`	number of swap outs
9	`ru_inblock`	block input operations
10	`ru_oublock`	block output operations
11	`ru_msgsnd`	messages sent
12	`ru_msgrcv`	messages received
13	`ru_nsignals`	signals received
14	`ru_nvcsw`	voluntary context switches
15	`ru_nivcsw`	involuntary context switches

This function will raise a `ValueError` if an invalid *who* parameter is specified. It may also raise `error` exception in unusual circumstances.

`resource.`**`getpagesize`**`()`

Returns the number of bytes in a system page. (This need not be the same as the hardware page size.)

The following `RUSAGE_*` symbols are passed to the `getrusage()` function to specify which processes information should be provided for.

`resource.`**`RUSAGE_SELF`**

Pass to `getrusage()` to request resources consumed by the calling process, which is the sum of resources used by all threads in the process.

`resource.`**`RUSAGE_CHILDREN`**

Pass to `getrusage()` to request resources consumed by child processes of the calling process which have been terminated and waited for.

`resource.`**`RUSAGE_BOTH`**

Pass to `getrusage()` to request resources consumed by both the current process and child processes. May not be available on all systems.

`resource.`**`RUSAGE_THREAD`**

Pass to `getrusage()` to request resources consumed by the current thread. May not be available on all systems.

New in version 3.2.