resource — System Resource Management

Purpose: Manage the system resource limits for a Unix program.

The functions in `resource` probe the current system resources consumed by a process, and place limits on them to control how much load a program can impose on a system.

Current Usage

Use `getrusage()` to probe the resources used by the current process and/or its children. The return value is a data structure containing several resource metrics based on the current state of the system.

Note

Not all of the resource values gathered are displayed here. Refer to the standard library documentation for `resource` for a more complete list.

```
# resource_getrusage.py

import resource
import time

RESOURCES = [
    ('ru_utime', 'User time'),
    ('ru_stime', 'System time'),
    ('ru_maxrss', 'Max. Resident Set Size'),
    ('ru_ixrss', 'Shared Memory Size'),
    ('ru_idrss', 'Unshared Memory Size'),
    ('ru_isrss', 'Stack Size'),
    ('ru_inblock', 'Block inputs'),
    ('ru_oublock', 'Block outputs'),
]

usage = resource.getrusage(resource.RUSAGE_SELF)

for name, desc in RESOURCES:
    print('{:<25} ({:<10}) = {}'.format(
        desc, name, getattr(usage, name)))
```

Because the test program is extremely simple, it does not use very many resources.

```
$ python3 resource_getrusage.py

User time                 (ru_utime  ) = 0.032299999999999995
System time               (ru_stime  ) = 0.01517
Max. Resident Set Size    (ru_maxrss ) = 9945088
Shared Memory Size        (ru_ixrss  ) = 0
Unshared Memory Size      (ru_idrss  ) = 0
Stack Size                (ru_isrss  ) = 0
Block inputs              (ru_inblock) = 0
Block outputs             (ru_oublock) = 0
```

Resource Limits

Separate from the current actual usage, it is possible to check the *limits* imposed on the application, and then change them.

```
# resource_getrlimit.py

import resource
```

```
LIMITS = [
    ('RLIMIT_CORE', 'core file size'),
    ('RLIMIT_CPU', 'CPU time'),
    ('RLIMIT_FSIZE', 'file size'),
    ('RLIMIT_DATA', 'heap size'),
    ('RLIMIT_STACK', 'stack size'),
    ('RLIMIT_RSS', 'resident set size'),
    ('RLIMIT_NPROC', 'number of processes'),
    ('RLIMIT_NOFILE', 'number of open files'),
    ('RLIMIT_MEMLOCK', 'lockable memory address'),
]

print('Resource limits (soft/hard):')
for name, desc in LIMITS:
    limit_num = getattr(resource, name)
    soft, hard = resource.getrlimit(limit_num)
    print('{:<23} {}/{}'.format(desc, soft, hard))
```

The return value for each limit is a tuple containing the *soft* limit imposed by the current configuration and the *hard* limit imposed by the operating system.

```
$ python3 resource_getrlimit.py

Resource limits (soft/hard):
core file size          0/9223372036854775807
CPU time                9223372036854775807/9223372036854775807
file size               9223372036854775807/9223372036854775807
heap size               9223372036854775807/9223372036854775807
stack size              8388608/67104768
resident set size       9223372036854775807/9223372036854775807
number of processes     1418/2128
number of open files    9472/9223372036854775807
lockable memory address 9223372036854775807/9223372036854775807
```

The limits can be changed with `setrlimit()`.

```
# resource_setrlimit_nofile.py

import resource
import os

soft, hard = resource.getrlimit(resource.RLIMIT_NOFILE)
print('Soft limit starts as  :', soft)

resource.setrlimit(resource.RLIMIT_NOFILE, (4, hard))

soft, hard = resource.getrlimit(resource.RLIMIT_NOFILE)
print('Soft limit changed to :', soft)

random = open('/dev/random', 'r')
print('random has fd =', random.fileno())
try:
    null = open('/dev/null', 'w')
except IOError as err:
    print(err)
else:
    print('null has fd =', null.fileno())
```

This example uses `RLIMIT_NOFILE` to control the number of open files allowed, changing it to a smaller soft limit than the default.

```
$ python3 resource_setrlimit_nofile.py

Soft limit starts as  : 9472
Soft limit changed to : 4
random has fd = 3
[Errno 24] Too many open files: '/dev/null'
```

It can also be useful to limit the amount of CPU time a process should consume, to avoid using too much. When the process runs past the allotted amount of time, it sent a `SIGXCPU` signal.

runs past the allotted amount of time, it sent a SIGXCPU signal.

```
# resource_setrlimit_cpu.py

import resource
import sys
import signal
import time


# Set up a signal handler to notify us
# when we run out of time.
def time_expired(n, stack):
    print('EXPIRED :', time.ctime())
    raise SystemExit('(time ran out)')


signal.signal(signal.SIGXCPU, time_expired)

# Adjust the CPU time limit
soft, hard = resource.getrlimit(resource.RLIMIT_CPU)
print('Soft limit starts as  :', soft)

resource.setrlimit(resource.RLIMIT_CPU, (1, hard))

soft, hard = resource.getrlimit(resource.RLIMIT_CPU)
print('Soft limit changed to :', soft)
print()

# Consume some CPU time in a pointless exercise
print('Starting:', time.ctime())
for i in range(200000):
    for i in range(200000):
        v = i * i

# We should never make it this far
print('Exiting :', time.ctime())
```

Normally the signal handler should flush all open files and close them, but in this case it just prints a message and exits.

```
$ python3 resource_setrlimit_cpu.py

Soft limit starts as  : 9223372036854775807
Soft limit changed to : 1

Starting: Sun Mar 18 16:21:52 2018
EXPIRED : Sun Mar 18 16:21:53 2018
(time ran out)
```

See also

- Standard library documentation for resource
- signal – For details on registering signal handlers.

platform — System Version Information

gc — Garbage Collector

Quick Links

Current Usage
Resource Limits

This page was last updated 2018-03-18.

Navigation

platform — System Version Information
gc — Garbage Collector

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example