NAME

getrlimit, **setrlimit** - control maximum system resource consumption

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

```
#include <sys/types.h>
#include <sys/time.h>
#include <sys/resource.h>

int
getrlimit(int resource, struct rlimit *rlp);

int
setrlimit(int resource, const struct rlimit *rlp);
```

DESCRIPTION

Limits on the consumption of system resources by the current process and each process it creates may be obtained with the **getrlimit**() system call, and set with the **setrlimit**() system call.

The *resource* argument is one of the following:

RLIMIT_AS The maximum amount (in bytes) of virtual memory the process is allowed to map.

RLIMIT_CORE The largest size (in bytes) core(5) file that may be created.

RLIMIT_CPU The maximum amount of cpu time (in seconds) to be used by each process.

RLIMIT_DATA The maximum size (in bytes) of the data segment for a process; this defines how far a program may extend its break with the sbrk(2) function.

RLIMIT_FSIZE The largest size (in bytes) file that may be created.

RLIMIT_KQUEUES
The maximum number of kqueues this user id is allowed to create.

RLIMIT_MEMLOCK
The maximum size (in bytes) which a process may lock into memory using the mlock(2) system call.

RLIMIT_NOFILE The maximum number of open files for this process.

RLIMIT_NPROC The maximum number of simultaneous processes for this user id.

RLIMIT_NPTS The maximum number of pseudo-terminals this user id is allowed to create.

RLIMIT_RSS When there is memory pressure and swap is available, prioritize eviction of a process' resident pages beyond this amount (in bytes). When memory is not under pressure, this rlimit is effectively ignored. Even when there is memory pressure, the amount of available swap space and some sysctl settings like vm.swap_enabled and vm.swap_idle_enabled can affect what happens to processes that have exceeded this size.

Processes that exceed their set RLIMIT_RSS are not signalled or halted. The limit is merely a hint to the VM daemon to prefer to deactivate pages from processes that have exceeded their set RLIMIT_RSS.

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.

RLIMIT_STACK The maximum size (in bytes) of the stack segment for a process; this defines how far a program's stack segment may be extended. Stack extension is performed automatically by the system.

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.

RLIMIT_VMEM An alias for RLIMIT_AS.

A resource limit is specified as a soft limit and a hard limit. When a soft limit is exceeded, a process might or might not receive a signal. For example, signals are generated when the cpu time or file size is exceeded, but not if the address space or RSS limit is exceeded. A program that exceeds the soft limit is allowed to continue execution until it reaches the hard limit, or modifies its own resource limit. Even reaching the hard limit does not necessarily halt a process. For example, if the RSS hard limit is exceeded, nothing happens.

The *rlimit* structure is used to specify the hard and soft limits on a resource.

```
struct rlimit {
        rlim_t  rlim_cur; /* current (soft) limit */
        rlim_t  rlim_max;       /* maximum value for rlim_cur */
};
```

Only the super-user may raise the maximum limits. Other users may only alter *rlim_cur* within the range from 0 to *rlim_max* or (irreversibly) lower *rlim_max*.

An "infinite" value for a limit is defined as RLIM_INFINITY.

Because this information is stored in the per-process information, this system call must be executed directly by the shell if it is to affect all future processes created by the shell; **limit** is thus a built-in command to csh(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way: a brk(2) function fails if the data space limit is reached. When the stack limit is reached, the process receives a segmentation fault (SIGSEGV); if this signal is not caught by a handler using the signal stack, this signal will kill the process.

A file I/O operation that would create a file larger that the process' soft limit will cause the write to fail and a signal SIGXFSZ to be generated; this normally terminates the process, but may be caught. When the soft cpu time limit is exceeded, a SIGXCPU signal is sent to the offending process.

When most operations would allocate more virtual memory than allowed by the soft limit of RLIMIT_AS, the operation fails with ENOMEM and no signal is raised. A notable exception is stack extension, described above. If stack extension would allocate more virtual memory than allowed by the soft limit of RLIMIT_AS, a SIGSEGV signal will be delivered. The caller is free to raise the soft address space limit up to the hard limit and retry the allocation.

RETURN VALUES

Upon successful completion, the value 0 is returned; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

The **getrlimit**() and **setrlimit**() system calls will fail if:

[EFAULT] The address specified for *rlp* is invalid.

[EPERM] The limit specified to **setrlimit**() would have raised the maximum limit value, and the caller is not the super-user.

SEE ALSO

csh(1), quota(1), quotactl(2), sigaction(2), sigaltstack(2), sysctl(3), ulimit(3)

HISTORY

The **getrlimit**() system call appeared in 4.2BSD.