NAME

getrlimit, setrlimit – control maximum system resource consumption

SYNOPSIS

#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 a variety of system resources by a process and each process it creates may be obtained with **getrlimit( )** and set with **setrlimit( )**.

Each call to either **getrlimit( )** or **setrlimit( )** identifies a specific resource to be operated upon as well as a resource limit. A resource limit is a pair of values: one specifying the current (soft) limit, the other a maximum (hard) limit. Soft limits may be changed by a process to any value that is less than or equal to the hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or equal to the soft limit. Only a process with an effective user ID of super-user can raise a hard limit. Both hard and soft limits can be changed in a single call to **setrlimit( )** subject to the constraints described above. Limits may have an "infinite" value of **RLIM_INFINITY**. *rlp* is a pointer to **struct rlimit** that includes the following members:

```
rlim_t    rlim_cur;    /* current (soft) limit */
rlim_t    rlim_max;    /* hard limit */
```

rlim_t is an arithmetic data type to which objects of type **int**, **size_t**, and **off_t** can be cast without loss of information.

The possible resources, their descriptions, and the actions taken when the current limit is exceeded are summarized in the table below:

RLIMIT_CORE The maximum size of a core file in bytes that may be created by a process. A limit of **0** will prevent the creation of a core file.

The writing of a core file will terminate at this size.

RLIMIT_CPU The maximum amount of CPU time in seconds used by a process. This is a soft limit only.

SIGXCPU is sent to the process. If the process is holding or ignoring **SIGXCPU**, the behavior is scheduling class defined.

RLIMIT_DATA The maximum size of a process's heap in bytes.

brk(2) will fail with **errno** set to **ENOMEM.**

RLIMIT_FSIZE The maximum size of a file in bytes that may be created by a process. A limit of **0** will prevent the creation of a file.

SIGXFSZ is sent to the process. If the process is holding or ignoring **SIGXFSZ**, continued attempts to increase the size of a file beyond the limit will fail with **errno** set to **EFBIG.**

RLIMIT_NOFILE One more than the maximum value that the system may assign to a newly created descriptor. This limit constrains the number of file descriptors that a process may create.

RLIMIT_STACK The maximum size of a process's stack in bytes. The system will not automatically grow the stack beyond this limit.

SIGSEGV is sent to the process. If the process is holding or ignoring **SIGSEGV**, or is catching **SIGSEGV** and has not made arrangements to use an alternate stack (see **sigaltstack**(2)), the disposition of **SIGSEGV** will be set to **SIG_DFL** before it is sent.

RLIMIT_VMEM The maximum size of a process's mapped address space in bytes.

brk(2) and **mmap**(2) functions will fail with **errno** set to **ENOMEM.** In addition, the automatic stack growth will fail with the effects outlined above.

Because limit information is stored in the per-process information, the shell builtin **ulimit** command must directly execute this system call if it is to affect all future processes created by the shell.

The value of the current limit of the following resources affect these implementation defined parameters:

Limit	Implementation Defined Constant
RLIMIT_FSIZE	**FCHR_MAX**
RLIMIT_NOFILE	**OPEN_MAX**

RETURN VALUES

Upon successful completion, the function **getrlimit( )** returns a value of 0; otherwise, it returns a value of –1 and sets **errno** to indicate an error.

ERRORS

Under the following conditions, the functions **getrlimit( )** and **setrlimit( )** fail and set **errno** to:

EFAULT *rlp* points to an illegal address.

EINVAL An invalid *resource* was specified; or in a **setrlimit( )** call, the new **rlim_cur** exceeds the new **rlim_max**.

EPERM The limit specified to **setrlimit( )** would have raised the maximum limit value, and the effective user of the calling process is not super-user.

SEE ALSO

brk(2), **open**(2), **sigaltstack**(2), **malloc**(3C), **signal**(3C), **signal**(5)

NOTES

RLIMIT_STACK:

Within a process **setrlimit()**, will increase the limit on the size of your stack, but will not move current memory segments to allow for that growth. Therefore, to guarantee that the process stack can grow to the limit, you must alter the limit prior to the execution of the process in which the new stack size is to be used.