NAME

intro − introduction to system calls

DESCRIPTION

Section 2 of the manual describes the Linux system calls. A system call is an entry point into the Linux kernel. Usually, system calls are not invoked directly: instead, most system calls have corresponding C library wrapper functions which perform the steps required (e.g., trapping to kernel mode) in order to invoke the system call. Thus, making a system call looks the same as invoking a normal library function.

In many cases, the C library wrapper function does nothing more than:

- copying arguments and the unique system call number to the registers where the kernel expects them;
- trapping to kernel mode, at which point the kernel does the real work of the system call;
- setting *errno* if the system call returns an error number when the kernel returns the CPU to user mode.

However, in a few cases, a wrapper function may do rather more than this, for example, performing some preprocessing of the arguments before trapping to kernel mode, or postprocessing of values returned by the system call. Where this is the case, the manual pages in Section 2 generally try to note the details of both the (usually GNU) C library API interface and the raw system call. Most commonly, the main DESCRIPTION will focus on the C library interface, and differences for the system call are covered in the NOTES section.

For a list of the Linux system calls, see **syscalls**(2).

RETURN VALUE

On error, most system calls return a negative error number (i.e., the negated value of one of the constants described in **errno**(3)). The C library wrapper hides this detail from the caller: when a system call returns a negative value, the wrapper copies the absolute value into the *errno* variable, and returns −1 as the return value of the wrapper.

The value returned by a successful system call depends on the call. Many system calls return 0 on success, but some can return nonzero values from a successful call. The details are described in the individual manual pages.

In some cases, the programmer must define a feature test macro in order to obtain the declaration of a system call from the header file specified in the man page SYNOPSIS section. (Where required, these feature test macros must be defined before including *any* header files.) In such cases, the required macro is described in the man page. For further information on feature test macros, see **feature_test_macros**(7).

STANDARDS

Certain terms and abbreviations are used to indicate UNIX variants and standards to which calls in this section conform. See **standards**(7).

NOTES

Calling directly

In most cases, it is unnecessary to invoke a system call directly, but there are times when the Standard C library does not implement a nice wrapper function for you. In this case, the programmer must manually invoke the system call using **syscall**(2). Historically, this was also possible using one of the _syscall macros described in **_syscall**(2).

Authors and copyright conditions

Look at the header of the manual page source for the author(s) and copyright conditions. Note that these can be different from page to page!

SEE ALSO

_syscall(2), **syscall**(2), **syscalls**(2), **errno**(3), **intro**(3), **capabilities**(7), **credentials**(7), **feature_test_macros**(7), **mq_overview**(7), **path_resolution**(7), **pipe**(7), **pty**(7), **sem_overview**(7), **shm_overview**(7), **signal**(7), **socket**(7), **standards**(7), **symlink**(7), **system_data_types**(7), **sysvipc**(7), **time**(7)