NAME

sigreturn, rt_sigreturn − return from signal handler and cleanup stack frame

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

int sigreturn(...);

DESCRIPTION

If the Linux kernel determines that an unblocked signal is pending for a process, then, at the next transition back to user mode in that process (e.g., upon return from a system call or when the process is rescheduled onto the CPU), it creates a new frame on the user-space stack where it saves various pieces of process context (processor status word, registers, signal mask, and signal stack settings).

The kernel also arranges that, during the transition back to user mode, the signal handler is called, and that, upon return from the handler, control passes to a piece of user-space code commonly called the "signal trampoline". The signal trampoline code in turn calls**sigr eturn**().

This **sigreturn**() call undoes everything that was done—changing the process's signal mask, switching signal stacks (see **sigaltstack**(2))—in order to invoke the signal handler. Using the information that was earlier saved on the user-space stack **sigreturn**() restores the process's signal mask, switches stacks, and restores the process's context (processor flags and registers, including the stack pointer and instruction pointer), so that the process resumes execution at the point where it was interrupted by the signal.

RETURN VALUE

sigreturn() never returns.

STANDARDS

Many UNIX-type systems have a **sigreturn**() system call or near equivalent. However, this call is not specified in POSIX, and details of its behavior vary across systems.

NOTES

sigreturn() exists only to allow the implementation of signal handlers. It should **never** be called directly. (Indeed, a simple **sigreturn**() wrapper in the GNU C library simply returns −1, with *errno* set to **ENOSYS**.) Details of the arguments (if any) passed to **sigreturn**() vary depending on the architecture. (On some architectures, such as x86-64, **sigreturn**() takes no arguments, since all of the information that it requires is available in the stack frame that was previously created by the kernel on the user-space stack.)

Once upon a time, UNIX systems placed the signal trampoline code onto the user stack. Nowadays, pages of the user stack are protected so as to disallow code execution. Thus, on contemporary Linux systems, depending on the architecture, the signal trampoline code lives either in the **vdso**(7) or in the C library. In the latter case, the C library's **sigaction**(2) wrapper function informs the kernel of the location of the trampoline code by placing its address in the *sa_restorer* field of the *sigaction* structure, and sets the **SA_RESTORER** flag in the *sa_flags* field.

The saved process context information is placed in a *ucontext_t* structure (see *<sys/ucontext.h>*). That structure is visible within the signal handler as the third argument of a handler established via **sigaction**(2) with the **SA_SIGINFO** flag.

On some other UNIX systems, the operation of the signal trampoline differs a little. In particular, on some systems, upon transitioning back to user mode, the kernel passes control to the trampoline (rather than the signal handler), and the trampoline code calls the signal handler (and then calls **sigreturn**() once the handler returns).

C library/kernel differences

The original Linux system call was named **sigreturn**(). However, with the addition of real-time signals in Linux 2.2, a new system call, **rt_sigreturn**() was added to support an enlarged *sigset_t* type. The GNU C library hides these details from us, transparently employing **rt_sigreturn**() when the kernel provides it.

SEE ALSO

kill(2), **restart_syscall**(2), **sigaltstack**(2), **signal**(2), **getcontext**(3), **signal**(7), **vdso**(7)