NAME

system - execute a shell command

SYNOPSIS

#include <stdlib.h>

int system(const char ****command*);**

DESCRIPTION

The **system**() library function uses fork(2) to create a child process that executes the shell command specified in *command* using execl(3) as follows:

```
execl("/bin/sh", "sh", "-c", command, (char *) 0);
```

system() returns after the command has been completed.

During execution of the command, **SIGCHLD** will be blocked, and **SIGINT** and **SIGQUIT** will be ignored, in the process that calls **system**() (these signals will be handled according to their defaults inside the child process that executes *command*).

If *command* is NULL, then **system**() returns a status indicating whether a shell is available on the system

RETURN VALUE

The return value of **system**() is one of the following:

* If *command* is NULL, then a nonzero value if a shell is available, or 0 if no shell is available.
* If a child process could not be created, or its status could not be retrieved, the return value is -1.
* If a shell could not be executed in the child process, then the return value is as though the child shell terminated by calling _exit(2) with the status 127.
* If all system calls succeed, then the return value is the termination status of the child shell used to execute *command*. (The termination status of a shell is the termination status of the last command it executes.)

In the last two cases, the return value is a "wait status" that can be examined using the macros described in waitpid(2). (i.e.,**WIFEXITED**(), **WEXITST ATUS**(), and so on).

system() does not affect the wait status of any other children.

ATTRIBUTES

For an explanation of the terms used in this section, see attributes(7).

Interface	**Attribute**	**Value**
system()	Thread safety	MT-Safe

CONFORMING TO

POSIX.1-2001, POSIX.1-2008, C89, C99.

NOTES

system() provides simplicity and convenience: it handles all of the details of calling fork(2), execl(3), and waitpid(2), as well as the necessary manipulations of signals; in addition, the shell performs the usual substitutions and I/O redirections for *command*. The main cost of**system**() is inef ficiency: additional system calls are required to create the process that runs the shell and to execute the shell.

If the **_XOPEN_SOURCE** feature test macro is defined (before including *any* header files), then the macros described in waitpid(2) (**WEXITSTATUS**(), etc.) are made available when including <*stdlib.h*>.

As mentioned, **system**() ignores **SIGINT** and **SIGQUIT**. This may make programs that call it from a loop uninterruptible, unless they take care themselves to check the exit status of the child. For example:

```
while (something) {
int ret = system("foo");

if (WIFSIGNALED(ret) &&
(WTERMSIG(ret) == SIGINT || WTERMSIG(ret) == SIGQUIT))
break;
}
```

Do not use **system**() from a program with set-user-ID or set-group-ID privileges, because strange values for some environment variables might be used to subvert system integrity. Use the exec(3) family of functions instead, but not execlp(3) or execvp(3). **system**() will not, in fact, work properly from programs with set-user-ID or set-group-ID privileges on systems on which */bin/sh* is bash version 2, since bash 2 drops privileges on startup. (Debian uses a modified bash which does not do this when invoked as **sh**.)

According to POSIX.1, it is unspecified whether handlers registered using **pthread_atfork(3)** are called during the execution of **system**(). In the glibc implementation, such handlers are not called.

In versions of glibc before 2.1.3, the check for the availability of */bin/sh* was not actually performed if *command* was NULL; instead it was always assumed to be available, and **system**() always returned 1 in this case. Since glibc 2.1.3, this check is performed because, even though POSIX.1-2001 requires a conforming implementation to provide a shell, that shell may not be available or executable if the calling program has previously called chroot(2) (which is not specified by POSIX.1-2001).

It is possible for the shell command to terminate with a status of 127, which yields a **system**() return value that is indistinguishable from the case where a shell could not be executed in the child process.

SEE ALSO

sh(1), execve(2), fork(2), sigaction(2), sigprocmask(2), wait(2), exec(3), signal(7)

COLOPHON

This page is part of release 4.10 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.