system(3) Library Functions Manual system(3)

NAME

system - execute a shell command

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <stdlib.h>

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

DESCRIPTION

The system() library function behaves as if it used fork(2) to create a child process that executed the shell command specified in command us? ing execl(3) as follows:

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

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().

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 and errno is set to indicate the error.

? If a shell could not be executed in the child process, then the re? turn 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 termina? tion status of the child shell used to execute command. (The termi? nation status of a shell is the termination status of the last com? mand it executes.)

examined using the macros described in waitpid(2). (i.e., WIFEXITED(), WEXITSTATUS(), and so on).

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

ERRORS

system() can fail with any of the same errors as fork(2).

ATTRIBUTES

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

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

STANDARDS

HISTORY

POSIX.1-2001, C89.

NOTES

system() provides simplicity and convenience: it handles all of the de? tails of calling fork(2), execl(3), and waitpid(2), as well as the nec? essary manipulations of signals; in addition, the shell performs the usual substitutions and I/O redirections for command. The main cost of system() is inefficiency: additional system calls are required to cre? ate 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) (WEXITSTA? TUS(), etc.) are made available when including <stdlib.h>.

As mentioned, system() ignores SIGINT and SIGQUIT. This may make pro? grams 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) &&
```

```
            break;
    }
```

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

Before glibc 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.

Caveats

Do not use system() from a privileged program (a set-user-ID or set- group-ID program, or a program with capabilities) because strange val? ues for some environment variables might be used to subvert system in? tegrity. For example, PATH could be manipulated so that an arbitrary

tions instead, but not execlp(3) or execvp(3) (which also use the PATH environment variable to search for an executable).

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 ver? sion 2: as a security measure, bash 2 drops privileges on startup. (Debian uses a different shell, dash(1), which does not do this when invoked as sh.)

Any user input that is employed as part of command should be carefully sanitized, to ensure that unexpected shell commands or command options are not executed. Such risks are especially grave when using system() from a privileged program.

BUGS

If the command name starts with a hyphen, sh(1) interprets the command name as an option, and the behavior is undefined. (See the -c option to sh(1).) To work around this problem, prepend the command with a space as in the following call:

```
system(" -unfortunate-command-name");
```

SEE ALSO

sh(1), execve(2), fork(2), sigaction(2), sigprocmask(2), wait(2),