NAME

execve − execute a file

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

```
#include <unistd.h>

int
execve(const char *path, char *const argv[], char *const envp[]);
```

DESCRIPTION

execve( ) transforms the calling process into a new process. The new process is constructed from an ordinary file, whose name is pointed to by *path*, called the *new process file*. This file is either an executable object file, or a file of data for an interpreter. An executable object file consists of an identifying header, followed by pages of data representing the initial program (text) and initialized data pages. Additional pages may be specified by the header to be initialized with zero data; see a.out(5).

An interpreter file begins with a line of the form:

#! *interpreter* [*arg*]

When an interpreter file is **execve**'d, the system **execve**'s runs the specified *interpreter*. If the optional *arg* is specified, it becomes the first argument to the *interpreter*, and the name of the originally **execve**'d file becomes the second argument; otherwise, the name of the originally **execve**'d file becomes the first argument. The original arguments are shifted over to become the subsequent arguments. The zeroth argument, normally the name of the **execve**'d file, is left unchanged.

The argument *argv* is a pointer to a null-terminated array of character pointers to null-terminated character strings. These strings construct the argument list to be made available to the new process. At least one argument must be present in the array; by custom, the first element should be the name of the executed program (for example, the last component of *path*).

The argument *envp* is also a pointer to a null-terminated array of character pointers to null-terminated strings. A pointer to this array is normally stored in the global variable *environ*. These strings pass information to the new process that is not directly an argument to the command (see environ(7)).

File descriptors open in the calling process image remain open in the new process image, except for those for which the close-on-exec flag is set (see close(2) and fcntl(2)). Descriptors that remain open are unaffected by **execve**().

Signals set to be ignored in the calling process are set to be ignored in the new process. Signals which are set to be caught in the calling process image are set to default action in the new process image. Blocked signals remain blocked regardless of changes to the signal action. The signal stack is reset to be undefined (see sigaction(2) for more information).

If the set-user-ID mode bit of the new process image file is set (see chmod(2)), the effective user ID of the new process image is set to the owner ID of the new process image file. If the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. (The effective group ID is the first element of the group list.) The real user ID, real group ID and other group IDs of the new process image remain the same as the calling process image. After any set-user-ID and set-group-ID processing, the effective user ID is recorded as the saved set-user-ID, and the effective group ID is recorded as the saved set-group-ID. These values may be used in changing the ef-

fective IDs later (see `setuid`(2)).

The new process also inherits the following attributes from the calling process:

process ID	see `getpid`(2)
parent process ID	see `getppid`(2)
process group ID	see `getpgrp`(2)
access groups	see `getgroups`(2)
working directory	see `chdir`(2)
root directory	see `chroot`(2)
control terminal	see `termios`(4)
resource usages	see `getrusage`(2)
interval timers	see `getitimer`(2)
resource limits	see `getrlimit`(2)
file mode mask	see `umask`(2)
signal mask	see `sigaction`(2), `sigprocmask`(2)

When a program is executed as a result of an **`execve`**( ) call, it is entered as follows:

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

where *`argc`* is the number of elements in *`argv`* (the ''arg count'') and *`argv`* points to the array of character pointers to the arguments themselves.

RETURN VALUES

As the **`execve`**( ) function overlays the current process image with a new process image the successful call has no process to return to. If **`execve`**( ) does return to the calling process an error has occurred; the return value will be -1 and the global variable *errno* is set to indicate the error.

ERRORS

`execve`( ) will fail and return to the calling process if:

[`ENOTDIR`] A component of the path prefix is not a directory.

[`ENAMETOOLONG`]
A component of a pathname exceeded {`NAME_MAX`} characters, or an entire path name exceeded {`PATH_MAX`} characters.

[`ENOENT`] The new process file does not exist.

[`ELOOP`] Too many symbolic links were encountered in translating the pathname.

[`EACCES`] Search permission is denied for a component of the path prefix, the new process file is not an ordinary file, it's file mode denies execute permission, or it is on a filesystem mounted with execution disabled (`MNT_NOEXEC` in <`sys/mount.h`>).

[`ENOEXEC`] The new process file has the appropriate access permission, but has an invalid magic number in its header.

[`ETXTBSY`] The new process file is a pure procedure (shared text) file that is currently open for writing or

reading by some process.

[ENOMEM] The new process requires more virtual memory than is allowed by the imposed maximum (getrlimit(2)).

[E2BIG] The number of bytes in the new process's argument list is larger than the system-imposed limit. The limit in the system as released is 262144 bytes (NCARGS in <sys/param.h>).

[EFAULT] The new process file is not as long as indicated by the size values in its header.

[EFAULT] *path*, *argv*, or *envp* point to an illegal address.

[EIO] An I/O error occurred while reading from the file system.

CAVEAT

If a program is *setuid* to a non-super-user, but is executed when the real *uid* is ''root'', then the program has some of the powers of a super-user as well.

SEE ALSO

_exit(2), fork(2), execl(3), environ(7)

STANDARDS

The **execve**() function conforms to IEEE Std 1003.1-1990 (''POSIX'').

HISTORY

The **execve**() function call appeared in 4.2BSD.