NAME

execl, **execlp**, **execle**, **exect**, **execv**, **execvp**, **execvpe**, **execvP** - execute a file

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

```
#include <unistd.h>

extern char **environ;

int
execl(const char *path, const char *arg, ..., NULL);

int
execlp(const char *file, const char *arg, ..., NULL);

int
execle(const char *path, const char *arg, ..., NULL, char *const envp[]);

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

int
execv(const char *path, char *const argv[]);

int
execvp(const char *file, char *const argv[]);

int
execvpe(const char *file, char *const argv[], char *const envp[]);

int
execvP(const char *file, const char *search_path, char *const argv[]);
```

DESCRIPTION

The **exec** family of functions replaces the current process image with a new process image. The functions described in this manual page are front-ends for the function execve(2). (See the manual page for execve(2) for detailed information about the replacement of the current process.)

The initial argument for these functions is the pathname of a file which is to be executed.

The *const char *arg* and subsequent ellipses in the **execl**(), **execlp**(), and **execle**() functions can be thought of as *arg0*, *arg1*, ..., *argn*. Together they describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program. The first argument, by convention, should point to the file name associated with the file being executed. The list of arguments *must* be terminated by a NULL pointer.

The **exect**(), **execv**(), **execvp**(), **execvpe**(), and **execvP**() functions provide an array of pointers to null-terminated strings that represent the argument list available to the new program. The first argument, by convention, should point to the file name associated with the file being executed. The array of pointers **must** be terminated by a NULL pointer.

The **execle**(), **exect**(), and **execvpe**() functions also specify the environment of the executed process by following the NULL pointer that terminates the list of arguments in the argument list or the pointer to the argv array with an additional argument. This additional argument is an array of pointers to null-terminated strings and *must* be terminated by a NULL pointer. The other functions take the environment for the new process image from the external variable *environ* in the current process.

Some of these functions have special semantics.

The functions **execlp**(), **execvp**(), **execvpe**(), and **execvP**() will duplicate the actions of the shell in searching for an executable file if the specified file name does not contain a slash "/" character. For **execlp**() and **execvp**(), **execvpe**(), search path is the path specified in the environment by "PATH" variable. If this variable is not specified, the default path is set according to the _PATH_DEFPATH definition in <*paths.h*>, which is set to "/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin". For **execvP**(), the search path is specified as an argument to the function. In addition, certain errors are treated specially.

If an error is ambiguous (for simplicity, we shall consider all errors except ENOEXEC as being ambiguous here, although only the critical error EACCES is really ambiguous), then these functions will act as if they stat the file to determine whether the file exists and has suitable execute permissions. If it does, they will return immediately with the global variable *errno* restored to the value set by **execve**(). Otherwise, the search will be continued. If the search completes without performing a successful **execve**() or terminating due to an error, these functions will return with the global variable *errno* set to EACCES or ENOENT according to whether at least one file with suitable execute permissions was found.

If the header of a file is not recognized (the attempted **execve**() returned ENOEXEC), these functions will execute the shell with the path of the file as its first argument. (If this attempt fails, no further

searching is done.)

The function **exect**() executes a file with the program tracing facilities enabled (see ptrace(2)).

RETURN VALUES

If any of the **exec**() functions returns, an error will have occurred. The return value is -1, and the global variable *errno* will be set to indicate the error.

FILES

/bin/sh The shell.

COMPATIBILITY

Historically, the default path for the **execlp**() and **execvp**() functions was "*:/bin:/usr/bin*". This was changed to remove the current directory to enhance system security.

The behavior of **execlp**() and **execvp**() when errors occur while attempting to execute the file is not quite historic practice, and has not traditionally been documented and is not specified by the POSIX standard.

Traditionally, the functions **execlp**() and **execvp**() ignored all errors except for the ones described above and ETXTBSY, upon which they retried after sleeping for several seconds, and ENOMEM and E2BIG, upon which they returned. They now return for ETXTBSY, and determine existence and executability more carefully. In particular, EACCES for inaccessible directories in the path prefix is no longer confused with EACCES for files with unsuitable execute permissions. In 4.4BSD, they returned upon all errors except EACCES, ENOENT, ENOEXEC and ETXTBSY. This was inferior to the traditional error handling, since it breaks the ignoring of errors for path prefixes and only improves the handling of the unusual ambiguous error EFAULT and the unusual error EIO. The behaviour was changed to match the behaviour of sh(1).

ERRORS

The **execl**(), **execle**(), **execlp**(), **execvp**(), **execvpe**(), and **execvP**() functions may fail and set *errno* for any of the errors specified for the library functions execve(2) and malloc(3).

The **exect**() and **execv**() functions may fail and set *errno* for any of the errors specified for the library function execve(2).

SEE ALSO

sh(1), execve(2), fork(2), ktrace(2), ptrace(2), environ(7)

STANDARDS

The **execl**(), **execv**(), **execle**(), **execlp**() and **execvp**() functions conform to IEEE Std 1003.1-1988

("POSIX.1"). The **execvpe**() function is a GNU extension.

HISTORY

The **exec**() function appeared in Version 1 AT&T UNIX. The **execl**() and **execv**() functions appeared in Version 2 AT&T UNIX. The **execlp**(), **execle**(), **execve**(), and **execvp**() functions appeared in Version 7 AT&T UNIX. The **execvP**() function first appeared in FreeBSD 5.2. The **execvpe**() function first appeared in FreeBSD 15.0.

BUGS

The type of the *argv* and *envp* parameters to **execle**(), **exect**(), **execv**(), **execvp**(), **execvpe**(), and **execvP**() is a historical accident and no sane implementation should modify the provided strings. The bogus parameter types trigger false positives from const correctness analyzers. On FreeBSD, the **__DECONST**() macro may be used to work around this limitation.

Due to a fluke of the C standard, on platforms other than FreeBSD the definition of NULL may be the untyped number zero, rather than a *(void *)0* expression. To distinguish the concepts, they are referred to as a "null pointer constant" and a "null pointer", respectively. On exotic computer architectures that FreeBSD does not support, the null pointer constant and null pointer may have a different representation. In general, where this document and others reference a NULL value, they actually imply a null pointer. E.g., for portability to non-FreeBSD operating systems on exotic computer architectures, one may use (char *)NULL in place of NULL when invoking **execl**(), **execle**(), and **execlp**().