NAME

execl, execv, execle, execve, execlp, execvp − execute a file

SYNOPSIS

```
int execl (name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;

int execv (name, argv)
char *name, *argv[ ];

int execle (name, arg0, arg1, ..., argn, 0, envp)
char *name, *arg0, *arg1, ..., *argn, *envp[ ];

int execve (name, argv, envp)
char *name, *argv[ ], *envp[ ];

int execlp (name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;

int execvp (name, argv)
char *name, *argv[ ];
```

DESCRIPTION

Exec in all its forms overlays the calling process with the named file, then transfers to the entry point of the core image of the file. There can be no return from a successful exec; the calling core image is lost.

File descriptors ordinarily remain open across *exec*, but may be requested to be automatically closed (see *ioctl*(2)). Ignored signals remain ignored across these calls, but signals that are caught (see *signal*(2)) are reset to their default values.

Each user has a *real* user ID and group ID and an *effective* user ID and group ID. The real ID identifies the person using the system; the effective ID determines his access privileges. *Exec* changes the effective user or group ID to the owner of the executed file if the file has the "set-user-ID" or "set-group-ID" modes. The real user and IDs are not affected.

The *name* argument is a pointer to the name of the file to be executed. The pointers *arg*[0], *arg*[1] ... address null-terminated strings. Conventionally *arg*[0] is the name of the file.

From C, two interfaces are available. *execl* is useful when a known file with known arguments is being called; the arguments to *execl* are the character strings constituting the file and the arguments; the first argument is conventionally the same as the file name (or its last component). A **0** argument must end the argument list.

The *execv* version is useful when the number of arguments is unknown in advance; the arguments to *execv* are the name of the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a 0 pointer.

When a C program is executed, it is called as follows:

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

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. As indicated, *argc* is conventionally at least one and the first member of the array points to a string containing the name of the file.

Argv is directly usable in another *execv* because *argv*[*argc*] is 0.

Envp is a pointer to an array of strings that constitute the *environment* of the process. Each string consists of a name, an =, and a null-terminated value. The array of pointers is terminated by a null pointer. The shell *sh*(1) passes an environment entry for each global shell variable defined when the program is called. See *environ*(7) for some conventionally used names. The C run-time start-off routine places a copy of *envp* in a global cell:

extern char **environ;

that is used by *execv* and *execl* to pass the environment to any subprograms executed by the current program. The *exec* routines use lower-level routines as follows to pass an environment explicitly:

```
execve (file, argv, environ);
execle (file, arg0, arg1, . . . , argn, 0, environ);
```

Execvp and *execlp* are called with the same arguments as *execv* and *execl,* but duplicate the Shell's actions in searching for an executable file in a list of directories. The directory list is obtained from the environment.

FILES

/bin/sh, or the value specified by the shell variable **$SHELL**, invoked if command file found by *execlp* or *execvp*

SEE ALSO

ioctl(2), fork(2), getenv(3C), environ(7)

DIAGNOSTICS

If the file cannot be found, if it is not executable, if it does not start with a valid magic number (see *a.out*(5)), if maximum memory is exceeded, if it is a pure-procedure program which is currently open for reading or writing, or if the arguments require too much space, a return constitutes the diagnostic; the return value is −1. Even for the super-user, at least one of the execute-permission bits must be set for a file to be executed.

ASSEMBLER

(exec = 11.)
sys exec; name; argv

(exece = 59.)
sys exece; name; argv; envp

Plain *exec* is replaced by *exece,* but remains for historical reasons.

When the called file starts execution, the stack pointer points to a word containing the number of arguments. Just above this number is a list of pointers to the argument strings, followed by a null pointer, followed by the pointers to the environment strings and then another null pointer. The strings themselves follow; a 0 word is left at the very top of memory.

```
 sp−>  nargs
       arg0
       ...
       argn
       0
       env0
       ...
       envm
       0
arg0:  <arg0\0>
       ...
env0:  <env0\0>
       0
```

This arrangement happens to conform well to C calling conventions.