System Calls exec(2)

NAME

exec, execl, execle, execlp, execv, execve, execvp - execute a file

SYNOPSIS

```
#include <unistd.h>

int execl(const char *path, const char *arg0, ..., const
char *argn, char * /*NULL*/);

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

int execle(const char *path, const char *arg0, ..., const
char *argn, char * /*NULL*/, char *const envp[]);

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

int execlp(const char *file, const char *arg0, ..., const
char *argn, char * /*NULL*/);

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

DESCRIPTION

Each of the functions in the exec family replaces the current process image with a new process image. The new image is constructed from a regular, executable file called the new process image file. This file is either an executable object file or a file of data for an interpreter. There is no return from a successful call to one of these functions because the calling process image is overlaid by the new process image.

An interpreter file begins with a line of the form

```
#! pathname [arg]
```

where pathname is the path of the interpreter, and arg is an optional argument. When an interpreter file is executed, the system invokes the specified interpreter. The pathname specified in the interpreter file is passed as arg0 to the interpreter. If arg was specified in the interpreter file, it is passed as arg1 to the interpreter. The remaining arguments to the interpreter are arg0 through argn of the originally exec'd file. The interpreter named by pathname must not be an interpreter file.

When a C-language program is executed as a result of this call, it is entered as a C-language function call as follows:

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

where argc is the argument count, argv is an array of character pointers to the arguments themselves, and envp is an array of character pointers to the environment strings. The argv and environ arrays are each terminated by a null pointer. The null pointer terminating the argv array is not counted in argc. The value of argc is non-negative, and if greater than 0, argv[0] points to a string containing the name of the file. If argc is 0, argv[0] is a null pointer, in which case there are no arguments. Applications should verify that argc is greater than 0 or that argv[0] is not a null pointer before dereferencing argv[0].

The arguments specified by a program with one of the exec functions are passed on to the new process image in the main() arguments.

The path argument points to a path name that identifies the new process image file.

The file argument is used to construct a pathname that identifies the new process image file . If the file argument contains a slash character, it is used as the pathname for this file. Otherwise, the path prefix for this file is

obtained by a search of the directories passed in the PATH environment variable (see environ(5)). The environment is supplied typically by the shell. If the process image file is not a valid executable object file, execlp() and execvp() use the contents of that file as standard input to the shell. In this case, the shell becomes the new process image. In a standard-conforming application (see standards(5)), the exec family of functions use /usr/xpg4/bin/sh (see ksh(1)); otherwise, they use /usr/bin/sh (see sh(1)).

The arguments represented by arg0... are pointers to null-terminated character strings. These strings constitute the argument list available to the new process image. The list is terminated by a null pointer. The arg0 argument should point to a filename that is associated with the process being started by one of the exec functions.

The argv argument is an array of character pointers to null-terminated strings. The last member of this array must be a null pointer. These strings constitute the argument list available to the new process image. The value in argv[0] should point to a filename that is associated with the process being started by one of the exec functions.

The envp argument is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The envp array is terminated by a null pointer. For execl(), execv(), execvp(), and execlp(), the C-language run-time start-off routine places a pointer to the environment of the calling process in the global object extern char **environ, and it is used to pass the environment of the calling process to the new process image.

The number of bytes available for the new process's combined argument and environment lists is ARG_MAX. It is implementation-dependent whether null terminators, pointers, and/or any alignment bytes are included in this total.

File descriptors open in the calling process image remain open in the new process image, except for those whose close-on-exec flag FD_CLOEXEC is set; (see fcntl(2)). For those file descriptors that remain open, all attributes of the open file description, including file locks, remain unchanged.

The preferred hardware address tranlation size (see memcntl(2)) for the stack and heap of the new process image are set to the default system page size.

Directory streams open in the calling process image are closed in the new process image.

The state of conversion descriptors and message catalogue descriptors in the new process image is undefined. For the new process, the equivalent of:

```
setlocale(LC_ALL, "C")
```

is executed at startup.

Signals set to the default action (SIG_DFL) in the calling process image are set to the default action in the new process image (see signal(3C)). Signals set to be ignored (SIG_IGN) by the calling process image are set to be ignored by the new process image. Signals set to be caught by the calling process image are set to the default action in the new process image (see signal(3HEAD)). After a successful call to any of the exec functions, alternate signal stacks are not preserved and the SA_ONSTACK flag is cleared for all signals.

After a successful call to any of the exec functions, any functions previously registered by atexit(3C) are no longer registered.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft resource limits.

If the ST_NOSUID bit is set for the file system containing the new process image file, then the effective user ID and effective group ID are unchanged in the new process image. 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. Similarly, 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 real user ID and real group ID of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID for use by setuid(2).

If the effective user-ID is root or superuser, the set-user-ID and set-group-ID bits will be honored when the process is being controlled by ptrace().

Any shared memory segments attached to the calling process image will not be attached to the new process image (see shmop(2)). Any mappings established through mmap() are not preserved across an exec. Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image. See mmap(2).

Memory locks established by the calling process via calls to mlockall(3C) or mlock(3C) are removed. If locked pages in the address space of the calling process are also mapped into the address spaces the locks established by the other processes will be unaffected by the call by this process to the exec function. If the exec function fails, the effect on memory locks is unspecified.

If _XOPEN_REALTIME is defined and has a value other than -1, any named semaphores open in the calling process are closed as if by appropriate calls to sem_close(3RT)

Profiling is disabled for the new process; see profil(2).

Timers created by the calling process with timer_create(3RT) are deleted before replacing the current process image with the new process image.

For the SCHED_FIFO and SCHED_RR scheduling policies, the policy and priority settings are not changed by a call to an exec function.

All open message queue descriptors in the calling process are closed, as described in mq_close(3RT).

Any outstanding asynchronous I/O operations may be cancelled. Those asynchronous I/O operations that are not canceled will complete as if the exec function had not yet occurred, but any associated signal notifications are suppressed. It is unspecified whether the exec function itself blocks awaiting such I/O completion. In no event, however, will the new process image created by the exec function be affected by the presence of outstanding asynchronous I/O operations at the time the exec function is called.

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

- nice value (see nice(2))
- scheduler class and priority (see priocntl(2))
- process ID
- parent process ID
- process group ID
- task ID
- supplementary group IDs
- semadj values (see semop(2))

- session membership (see exit(2) and signal(3C))
- real user ID
- real group ID
- project ID
- trace flag (see ptrace(2) request 0)
- time left until an alarm clock signal (see alarm(2))
- current working directory
- root directory
- file mode creation mask (see umask(2))
- file size limit (see ulimit(2))
- resource limits (see getrlimit(2))
- tms_utime, tms_stime, tms_cutime, and tms_cstime (see times(2))
- file-locks (see fcntl(2) and lockf(3C))
- controlling terminal
- process signal mask (see sigprocmask(2))
- pending signals (see sigpending(2))
- processor bindings (see processor_bind(2))
- processor set bindings (see pset_bind(2))

A call to any exec function from a process with more than one thread results in all threads being terminated and the new executable image being loaded and executed. No destructor functions will be called.

Upon successful completion, each of the functions in the exec family marks for update the st_atime field of the file. If an exec function failed but was able to locate the process image file, whether the st_atime field is marked for update is unspecified. Should the function succeed, the process image file is considered to have been opened with open(2). The corresponding close(2) is considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the exec functions. The argv[] and envp[] arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the exec functions, except as a consequence of replacing the process image.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft limits.

RETURN VALUES

If a function in the exec family returns to the calling process image, an error has occurred; the return value is -1 and errno is set to indicate the error.

ERRORS

The exec functions will fail if:

E2BIG The number of bytes in the new process's argument list is greater than the system-imposed limit of {ARG_MAX} bytes. The argument list limit is sum of the size of the argument list plus the size of the environment's exported shell variables.

EACCES

Search permission is denied for a directory listed in the new process file's path prefix; the new process file is not an ordinary file; or the new process file mode denies execute permission.

EAGAIN
Total amount of system memory available when reading using raw I/O is temporarily insufficient.

EFAULT
An argument points to an illegal address.

EINTR A signal was caught during the execution of one of the functions in the exec family.

ELOOP Too many symbolic links were encountered in translating path or file.

ENAMETOOLONG
The length of the file or path argument exceeds {PATH_MAX}, or the length of a file or path component exceeds {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect.

ENOENT
One or more components of the new process path name of the file do not exist or is a null pathname.

ENOLINK
The path argument points to a remote machine and the link to that machine is no longer active.

ENOTDIR
A component of the new process path of the file prefix is not a directory.

The exec functions, except for execlp() and execvp(), will fail if:

ENOEXEC
The new process image file has the appropriate access permission but is not in the proper format.

The exec functions may fail if:

ENAMETOOLONG
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

ENOMEM
The new process image requires more memory than is allowed by the hardware or system-imposed by memory management constraints. See brk(2).

ETXTBSY
The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.

USAGE
As the state of conversion descriptors and message catalogue descriptors in the new process image is undefined, portable applications should not rely on their use and should close them prior to calling one of the exec functions.

Applications that require other than the default POSIX locale should call setlocale(3C) with the appropriate parameters to establish the locale of thenew process.

The environ array should not be accessed directly by the application.

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Standard
MT-Level	execle() and execve() are Async-Signal-Safe

SEE ALSO

ksh(1), ps(1), sh(1), alarm(2), brk(2), chmod(2), exit(2), fcntl(2), fork(2), getrlimit(2), memcntl(2), mmap(2), nice(2), priocntl(2), profil(2), semop(2), shmop(2), sigpending(2), sigprocmask(2), times(2), umask(2), lockf(3C), ptrace(2), setlocale(3C), signal(3C), system(3C), timer_create(3RT), a.out(4), attributes(5), environ(5), standards(5)

WARNINGS

If a program is setuid to a user ID other than the superuser, and the program is executed when the real user ID is super-user, then the program has some of the powers of a super-user as well.

SunOS 5.9 Last change: 20 Dec 2001