## NAME

**posix_spawn**, **posix_spawnp** - spawn a process

## LIBRARY

Standard C Library (libc, -lc)

## SYNOPSIS

**#include <spawn.h>**

*int*
**posix_spawn**(*pid_t *restrict pid, const char *restrict path,*
*const posix_spawn_file_actions_t *file_actions, const posix_spawnattr_t *restrict attrp,*
*char *const argv[restrict], char *const envp[restrict]*);

*int*
**posix_spawnp**(*pid_t *restrict pid, const char *restrict file,*
*const posix_spawn_file_actions_t *file_actions, const posix_spawnattr_t *restrict attrp,*
*char *const argv[restrict], char *const envp[restrict]*);

## DESCRIPTION

The **posix_spawn**() and **posix_spawnp**() functions create a new process (child process) from the specified process image. The new process image is constructed from a regular executable file called the new process image file.

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

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

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the variable:

```
extern char **environ;
```

points to an array of character pointers to the environment strings.

The argument *argv* is an array of character pointers to null-terminated strings. The last member of this array is a null pointer and is not counted in *argc*. 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 image being started by the **posix_spawn**() or **posix_spawnp**() function.

The argument *envp* is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The environment array is terminated by a null pointer.

The *path* argument to **posix_spawn**() is a pathname that identifies the new process image file to execute.

The *file* parameter to **posix_spawnp**() is used to construct a pathname that identifies the new process image file. If the file parameter contains a slash character, the file parameter is used as the pathname for the new process image file. Otherwise, the path prefix for this file is obtained by a search of the directories passed as the environment variable "PATH". 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".

If *file_actions* is a null pointer, then file descriptors open in the calling process remain open in the child process, except for those whose close-on-exec flag FD_CLOEXEC is set (see **fcntl**()). For those file descriptors that remain open, all attributes of the corresponding open file descriptions, including file locks (see **fcntl**()), remain unchanged.

If *file_actions* is not NULL, then the file descriptors open in the child process are those open in the calling process as modified by the spawn file actions object pointed to by *file_actions* and the FD_CLOEXEC flag of each remaining open file descriptor after the spawn file actions have been processed. The effective order of processing the spawn file actions are:

1. The set of open file descriptors for the child process initially are the same set as is open for the calling process. All attributes of the corresponding open file descriptions, including file locks (see **fcntl**()), remain unchanged.

2. The signal mask, signal default actions, and the effective user and group IDs for the child process are changed as specified in the attributes object referenced by *attrp*.

3. The file actions specified by the spawn file actions object are performed in the order in which they were added to the spawn file actions object.

4. Any file descriptor that has its FD_CLOEXEC flag set (see **fcntl**()) is closed.

The *posix_spawnattr_t* spawn attributes object type is defined in <*spawn.h*>. It contains the attributes defined below.

If the POSIX_SPAWN_SETPGROUP flag is set in the spawn-flags attribute of the object referenced by *attrp*, and the spawn-pgroup attribute of the same object is non-zero, then the child’s process group is as specified in the spawn-pgroup attribute of the object referenced by *attrp*.

As a special case, if the POSIX_SPAWN_SETPGROUP flag is set in the spawn-flags attribute of the object referenced by *attrp*, and the spawn-pgroup attribute of the same object is set to zero, then the child is in a new process group with a process group ID equal to its process ID.

If the POSIX_SPAWN_SETPGROUP flag is not set in the spawn-flags attribute of the object referenced by *attrp*, the new child process inherits the parent's process group.

If the POSIX_SPAWN_SETSCHEDPARAM flag is set in the spawn-flags attribute of the object referenced by *attrp*, but POSIX_SPAWN_SETSCHEDULER is not set, the new process image initially has the scheduling policy of the calling process with the scheduling parameters specified in the spawn-schedparam attribute of the object referenced by *attrp*.

If the POSIX_SPAWN_SETSCHEDULER flag is set in the spawn-flags attribute of the object referenced by *attrp* (regardless of the setting of the POSIX_SPAWN_SETSCHEDPARAM flag), the new process image initially has the scheduling policy specified in the spawn-schedpolicy attribute of the object referenced by *attrp* and the scheduling parameters specified in the spawn-schedparam attribute of the same object.

The POSIX_SPAWN_RESETIDS flag in the spawn-flags attribute of the object referenced by *attrp* governs the effective user ID of the child process. If this flag is not set, the child process inherits the parent process' effective user ID. If this flag is set, the child process' effective user ID is reset to the parent's real user ID. In either case, if the set-user-ID mode bit of the new process image file is set, the effective user ID of the child process becomes that file's owner ID before the new process image begins execution.

The POSIX_SPAWN_RESETIDS flag in the spawn-flags attribute of the object referenced by *attrp* also governs the effective group ID of the child process. If this flag is not set, the child process inherits the parent process' effective group ID. If this flag is set, the child process' effective group ID is reset to the parent's real group ID. In either case, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the child process becomes that file's group ID before the new process image begins execution.

If the POSIX_SPAWN_SETSIGMASK flag is set in the spawn-flags attribute of the object referenced by *attrp*, the child process initially has the signal mask specified in the spawn-sigmask attribute of the object referenced by *attrp*.

If the POSIX_SPAWN_SETSIGDEF flag is set in the spawn-flags attribute of the object referenced by *attrp*, the signals specified in the spawn-sigdefault attribute of the same object are set to their default actions in the child process. Signals set to the default action in the parent process are set to the default action in the child process.

Signals set to be caught by the calling process are set to the default action in the child process.

Signals set to be ignored by the calling process image are set to be ignored by the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the spawn-flags attribute of the object referenced by *attrp* and the signals being indicated in the spawn-sigdefault attribute of the object referenced by *attrp*.

If the value of the *attrp* pointer is NULL, then the default values are used.

All process attributes, other than those influenced by the attributes set in the object referenced by *attrp* as specified above or by the file descriptor manipulations specified in *file_actions*, appear in the new process image as though **vfork**() had been called to create a child process and then **execve**() had been called by the child process to execute the new process image.

The implementation uses **vfork**(), thus the fork handlers are not run when **posix_spawn**() or **posix_spawnp**() is called.

## RETURN VALUES

Upon successful completion, **posix_spawn**() and **posix_spawnp**() return the process ID of the child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and return zero as the function return value. Otherwise, no child process is created, no value is stored into the variable pointed to by *pid*, and an error number is returned as the function return value to indicate the error. If the *pid* argument is a null pointer, the process ID of the child is not returned to the caller.

## ERRORS

1. If **posix_spawn**() and **posix_spawnp**() fail for any of the reasons that would cause **vfork**() or one of the **exec** to fail, an error value is returned as described by **vfork**() and **exec**, respectively (or, if the error occurs after the calling process successfully returns, the child process exits with exit status 127).

2. If **POSIX_SPAWN_SETPGROUP** is set in the spawn-flags attribute of the object referenced by attrp, and **posix_spawn**() or **posix_spawnp**() fails while changing the child's process group, an error value is returned as described by **setpgid**() (or, if the error occurs after the calling process successfully returns, the child process exits with exit status 127).

3. If **POSIX_SPAWN_SETSCHEDPARAM** is set and **POSIX_SPAWN_SETSCHEDULER** is not set in the spawn-flags attribute of the object referenced by attrp, then if **posix_spawn**() or **posix_spawnp**() fails for any of the reasons that would cause **sched_setparam**() to fail, an error value is returned as described by **sched_setparam**() (or, if the error occurs after the calling process successfully returns, the child process exits with exit status 127).

4. If **POSIX_SPAWN_SETSCHEDULER** is set in the spawn-flags attribute of the object referenced by attrp, and if **posix_spawn**() or **posix_spawnp**() fails for any of the reasons that would cause **sched_setscheduler**() to fail, an error value is returned as described by **sched_setscheduler**() (or, if the error occurs after the calling process successfully returns, the child process exits with exit status 127).

5. If the *file_actions* argument is not NULL, and specifies any dup2 or open actions to be performed, and if **posix_spawn**() or **posix_spawnp**() fails for any of the reasons that would cause **dup2**() or **open**() to fail, an error value is returned as described by **dup2**() and **open**(), respectively (or, if the error occurs after the calling process successfully returns, the child process exits with exit status 127). An open file action may, by itself, result in any of the errors described by **dup2**(), in addition to those described by **open**(). This implementation ignores any errors from **close**(), including trying to close a descriptor that is not open. The ignore extends to any errors from individual file descriptors **close**() executed as part of the **closefrom**() action.

## SEE ALSO

close(2), dup2(2), execve(2), fcntl(2), open(2), sched_setparam(2), sched_setscheduler(2), setpgid(2), vfork(2), posix_spawn_file_actions_addclose(3), posix_spawn_file_actions_addclosefrom_np(3), posix_spawn_file_actions_adddup2(3), posix_spawn_file_actions_addopen(3), posix_spawn_file_actions_addchdir_np(3), posix_spawn_file_actions_addfchdir_np(3), posix_spawn_file_actions_destroy(3), posix_spawn_file_actions_init(3), posix_spawnattr_destroy(3), posix_spawnattr_getflags(3), posix_spawnattr_getpgroup(3), posix_spawnattr_getschedparam(3), posix_spawnattr_getschedpolicy(3), posix_spawnattr_getsigdefault(3), posix_spawnattr_getsigmask(3), posix_spawnattr_init(3), posix_spawnattr_setflags(3), posix_spawnattr_setpgroup(3), posix_spawnattr_setschedparam(3), posix_spawnattr_setschedpolicy(3), posix_spawnattr_setsigdefault(3), posix_spawnattr_setsigmask(3)

## STANDARDS

The **posix_spawn**() and **posix_spawnp**() functions conform to IEEE Std 1003.1-2001 ("POSIX.1"), except that they ignore all errors from **close**(). A future update of the Standard is expected to require that these functions not fail because a file descriptor to be closed (via **posix_spawn_file_actions_addclose**()) is not open.

## HISTORY

The **posix_spawn**() and **posix_spawnp**() functions first appeared in FreeBSD 8.0.

## AUTHORS

Ed Schouten <*ed@FreeBSD.org*>