Rocky Enterprise Linux 9.2 Manual Pages on command 'posix_spawnp.3'

$ man posix_spawnp.3

POSIX_SPAWN(3) Linux Programmer's Manual POSIX_SPAWN(3)

NAME

posix_spawn, posix_spawnp - spawn a process

SYNOPSIS

```
#include <spawn.h>
int posix_spawn(pid_t *pid, const char *path,
                const posix_spawn_file_actions_t *file_actions,
                const posix_spawnattr_t *attrp,
                char *const argv[], char *const envp[]);
int posix_spawnp(pid_t *pid, const char *file,
                 const posix_spawn_file_actions_t *file_actions,
                 const posix_spawnattr_t *attrp,
                 char *const argv[], char *const envp[]);
```

DESCRIPTION

The posix_spawn() and posix_spawnp() functions are used to create a new child process that executes a specified file. These functions were specified by POSIX to provide a standardized method of creating new processes on machines that lack the capability to support the fork(2)

system call. These machines are generally small, embedded systems lacking MMU support.

The posix_spawn() and posix_spawnp() functions provide the functional? ity of a combined fork(2) and exec(3), with some optional housekeeping steps in the child process before the exec(3). These functions are not meant to replace the fork(2) and execve(2) system calls. In fact, they provide only a subset of the functionality that can be achieved by us? ing the system calls.

The only difference between posix_spawn() and posix_spawnp() is the manner in which they specify the file to be executed by the child process. With posix_spawn(), the executable file is specified as a pathname (which can be absolute or relative). With posix_spawnp(), the executable file is specified as a simple filename; the system searches for this file in the list of directories specified by PATH (in the same way as for execvp(3)). For the remainder of this page, the discussion is phrased in terms of posix_spawn(), with the understanding that posix_spawnp() differs only on the point just described.

The remaining arguments to these two functions are as follows:

* The pid argument points to a buffer that is used to return the process ID of the new child process.
* The file_actions argument points to a spawn file actions object that specifies file-related actions to be performed in the child between the fork(2) and exec(3) steps. This object is initialized and popu? lated before the posix_spawn() call using posix_spawn_file_ac? tions_init(3) and the posix_spawn_file_actions_*() functions.
* The attrp argument points to an attributes objects that specifies various attributes of the created child process. This object is initialized and populated before the posix_spawn() call using posix_spawnattr_init(3) and the posix_spawnattr_*() functions.
* The argv and envp arguments specify the argument list and environ? ment for the program that is executed in the child process, as for execve(2).

Below, the functions are described in terms of a three-step process:

the fork() step, the pre-exec() step (executed in the child), and the exec() step (executed in the child).

fork() step

Since glibc 2.24, the posix_spawn() function commences by calling clone(2) with CLONE_VM and CLONE_VFORK flags. Older implementations use fork(2), or possibly vfork(2) (see below).

The PID of the new child process is placed in *pid. The posix_spawn() function then returns control to the parent process.

Subsequently, the parent can use one of the system calls described in wait(2) to check the status of the child process. If the child fails in any of the housekeeping steps described below, or fails to execute the desired file, it exits with a status of 127.

Before glibc 2.24, the child process is created using vfork(2) instead of fork(2) when either of the following is true:

* the spawn-flags element of the attributes object pointed to by attrp contains the GNU-specific flag POSIX_SPAWN_USEVFORK; or
* file_actions is NULL and the spawn-flags element of the attributes object pointed to by attrp does not contain POSIX_SPAWN_SETSIGMASK, POSIX_SPAWN_SETSIGDEF, POSIX_SPAWN_SETSCHEDPARAM, POSIX_SPAWN_SETSCHEDULER, POSIX_SPAWN_SETPGROUP, or POSIX_SPAWN_RE? SETIDS.

In other words, vfork(2) is used if the caller requests it, or if there is no cleanup expected in the child before it exec(3)s the requested file.

pre-exec() step: housekeeping

In between the fork() and the exec() steps, a child process may need to perform a set of housekeeping actions. The posix_spawn() and posix_spawnp() functions support a small, well-defined set of system tasks that the child process can accomplish before it executes the exe? cutable file. These operations are controlled by the attributes object pointed to by attrp and the file actions object pointed to by file_ac? tions. In the child, processing is done in the following sequence:

1. Process attribute actions: signal mask, signal default handlers,

scheduling algorithm and parameters, process group, and effective user and group IDs are changed as specified by the attributes object pointed to by attrp.

2. File actions, as specified in the file_actions argument, are per? formed in the order that they were specified using calls to the posix_spawn_file_actions_add*() functions.

3. File descriptors with the FD_CLOEXEC flag set are closed.

All process attributes in the child, other than those affected by at? tributes specified in the object pointed to by attrp and the file ac? tions in the object pointed to by file_actions, will be affected as though the child was created with fork(2) and it executed the program with execve(2).

The process attributes actions are defined by the attributes object pointed to by attrp. The spawn-flags attribute (set using posix_spaw? nattr_setflags(3)) controls the general actions that occur, and other attributes in the object specify values to be used during those ac? tions.

The effects of the flags that may be specified in spawn-flags are as follows:

POSIX_SPAWN_SETSIGMASK

Set the signal mask to the signal set specified in the spawn-sigmask attribute of the object pointed to by attrp. If the POSIX_SPAWN_SETSIGMASK flag is not set, then the child inherits the parent's signal mask.

POSIX_SPAWN_SETSIGDEF

Reset the disposition of all signals in the set specified in the spawn-sigdefault attribute of the object pointed to by attrp to the default. For the treatment of the dispositions of signals not specified in the spawn-sigdefault attribute, or the treat? ment when POSIX_SPAWN_SETSIGDEF is not specified, see execve(2).

POSIX_SPAWN_SETSCHEDPARAM

If this flag is set, and the POSIX_SPAWN_SETSCHEDULER flag is not set, then set the scheduling parameters to the parameters

specified in the spawn-schedparam attribute of the object pointed to by attrp.

POSIX_SPAWN_SETSCHEDULER

Set the scheduling policy algorithm and parameters of the child, as follows:

* The scheduling policy is set to the value specified in the spawn-schedpolicy attribute of the object pointed to by at? trp.
* The scheduling parameters are set to the value specified in the spawn-schedparam attribute of the object pointed to by attrp (but see BUGS).

If the POSIX_SPAWN_SETSCHEDPARAM and POSIX_SPAWN_SETSCHEDPOLICY flags are not specified, the child inherits the corresponding scheduling attributes from the parent.

POSIX_SPAWN_RESETIDS

If this flag is set, reset the effective UID and GID to the real UID and GID of the parent process. If this flag is not set, then the child retains the effective UID and GID of the parent. In either case, if the set-user-ID and set-group-ID permission bits are enabled on the executable file, their effect will over? ride the setting of the effective UID and GID (se execve(2)).

POSIX_SPAWN_SETPGROUP

Set the process group to the value specified in the spawn-pgroup attribute of the object pointed to by attrp. If the spawn-pgroup attribute has the value 0, the child's process group ID is made the same as its process ID. If the POSIX_SPAWN_SETP? GROUP flag is not set, the child inherits the parent's process group ID.

POSIX_SPAWN_USEVFORK

Since glibc 2.24, this flag has no effect. On older implementa? tions, setting this flag forces the fork() step to use vfork(2) instead of fork(2). The _GNU_SOURCE feature test macro must be defined to obtain the definition of this constant.

POSIX_SPAWN_SETSID (since glibc 2.26)

If this flag is set, the child process shall create a new ses? sion and become the session leader. The child process shall also become the process group leader of the new process group in the session (see setsid(2)). The _GNU_SOURCE feature test macro must be defined to obtain the definition of this constant.

If attrp is NULL, then the default behaviors described above for each flag apply.

The file_actions argument specifies a sequence of file operations that are performed in the child process after the general processing de? scribed above, and before it performs the exec(3). If file_actions is NULL, then no special action is taken, and standard exec(3) semantics apply?file descriptors open before the exec remain open in the new process, except those for which the FD_CLOEXEC flag has been set. File locks remain in place.

If file_actions is not NULL, then it contains an ordered set of re? quests to open(2), close(2), and dup2(2) files. These requests are added to the file_actions by posix_spawn_file_actions_addopen(3), posix_spawn_file_actions_addclose(3), and posix_spawn_file_actions_ad? ddup2(3). The requested operations are performed in the order they were added to file_actions.

If any of the housekeeping actions fails (due to bogus values being passed or other reasons why signal handling, process scheduling, process group ID functions, and file descriptor operations might fail), the child process exits with exit value 127.

exec() step

Once the child has successfully forked and performed all requested pre-exec steps, the child runs the requested executable.

The child process takes its environment from the envp argument, which is interpreted as if it had been passed to execve(2). The arguments to the created process come from the argv argument, which is processed as for execve(2).

RETURN VALUE

Upon successful completion, posix_spawn() and posix_spawnp() place the PID of the child process in pid, and return 0. If there is an error during the fork() step, then no child is created, the contents of *pid are unspecified, and these functions return an error number as de? scribed below.

Even when these functions return a success status, the child process may still fail for a plethora of reasons related to its pre-exec() ini? tialization. In addition, the exec(3) may fail. In all of these cases, the child process will exit with the exit value of 127.

ERRORS

The posix_spawn() and posix_spawnp() functions fail only in the case where the underlying fork(2), vfork(2) or clone(2) call fails; in these cases, these functions return an error number, which will be one of the errors described for fork(2), vfork(2) or clone(2).

In addition, these functions fail if:

ENOSYS Function not supported on this system.

VERSIONS

The posix_spawn() and posix_spawnp() functions are available since glibc 2.2.

CONFORMING TO

POSIX.1-2001, POSIX.1-2008.

NOTES

The housekeeping activities in the child are controlled by the objects pointed to by attrp (for non-file actions) and file_actions In POSIX parlance, the posix_spawnattr_t and posix_spawn_file_actions_t data types are referred to as objects, and their elements are not specified by name. Portable programs should initialize these objects using only the POSIX-specified functions. (In other words, although these objects may be implemented as structures containing fields, portable programs must avoid dependence on such implementation details.)

According to POSIX, it is unspecified whether fork handlers established with pthread_atfork(3) are called when posix_spawn() is invoked. Since glibc 2.24, the fork handlers are not executed in any case. On older

implementations, fork handlers are called only if the child is created using fork(2).

There is no "posix_fspawn" function (i.e., a function that is to posix_spawn() as fexecve(3) is to execve(2)). However, this function? ality can be obtained by specifying the path argument as one of the files in the caller's /proc/self/fd directory.

BUGS

POSIX.1 says that when POSIX_SPAWN_SETSCHEDULER is specified in spawn-flags, then the POSIX_SPAWN_SETSCHEDPARAM (if present) is ignored. However, before glibc 2.14, calls to posix_spawn() failed with an error if POSIX_SPAWN_SETSCHEDULER was specified without also specifying POSIX_SPAWN_SETSCHEDPARAM.

EXAMPLES

The program below demonstrates the use of various functions in the POSIX spawn API. The program accepts command-line attributes that can be used to create file actions and attributes objects. The remaining command-line arguments are used as the executable name and command-line arguments of the program that is executed in the child.

In the first run, the date(1) command is executed in the child, and the posix_spawn() call employs no file actions or attributes objects.

```
$ ./a.out date
PID of child: 7634
Tue Feb  1 19:47:50 CEST 2011
Child status: exited, status=0
```

In the next run, the -c command-line option is used to create a file actions object that closes standard output in the child. Consequently, date(1) fails when trying to perform output and exits with a status of 1.

```
$ ./a.out -c date
PID of child: 7636
date: write error: Bad file descriptor
Child status: exited, status=1
```

In the next run, the -s command-line option is used to create an at?

tributes object that specifies that all (blockable) signals in the child should be blocked. Consequently, trying to kill child with the default signal sent by kill(1) (i.e., SIGTERM) fails, because that sig? nal is blocked. Therefore, to kill the child, SIGKILL is necessary (SIGKILL can't be blocked).

```
$ ./a.out -s sleep 60 &
[1] 7637
$ PID of child: 7638
$ kill 7638
$ kill -KILL 7638
$ Child status: killed by signal 9
[1]+  Done                    ./a.out -s sleep 60
```

When we try to execute a nonexistent command in the child, the exec(3) fails and the child exits with a status of 127.

```
$ ./a.out xxxxx
PID of child: 10190
Child status: exited, status=127
```

Program source

```
#include <spawn.h>
#include <stdint.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <wait.h>
#include <errno.h>
#define errExit(msg)    do { perror(msg); \
                         exit(EXIT_FAILURE); } while (0)
#define errExitEN(en, msg) \
                       do { errno = en; perror(msg); \
                         exit(EXIT_FAILURE); } while (0)
char **environ;
int
```

```
main(int argc, char *argv[])
{
    pid_t child_pid;
    int s, opt, status;
    sigset_t mask;
    posix_spawnattr_t attr;
    posix_spawnattr_t *attrp;
    posix_spawn_file_actions_t file_actions;
    posix_spawn_file_actions_t *file_actionsp;
    /* Parse command-line options, which can be used to specify an
       attributes object and file actions object for the child. */
    attrp = NULL;
    file_actionsp = NULL;
    while ((opt = getopt(argc, argv, "sc")) != -1) {
        switch (opt) {
        case 'c':       /* -c: close standard output in child */
            /* Create a file actions object and add a "close"
               action to it */
            s = posix_spawn_file_actions_init(&file_actions);
            if (s != 0)
                errExitEN(s, "posix_spawn_file_actions_init");
            s = posix_spawn_file_actions_addclose(&file_actions,
                                        STDOUT_FILENO);
            if (s != 0)
                errExitEN(s, "posix_spawn_file_actions_addclose");
            file_actionsp = &file_actions;
            break;
        case 's':       /* -s: block all signals in child */
            /* Create an attributes object and add a "set signal mask"
               action to it */
            s = posix_spawnattr_init(&attr);
            if (s != 0)
                errExitEN(s, "posix_spawnattr_init");
```

```
            s = posix_spawnattr_setflags(&attr, POSIX_SPAWN_SETSIGMASK);
            if (s != 0)
                errExitEN(s, "posix_spawnattr_setflags");
            sigfillset(&mask);
            s = posix_spawnattr_setsigmask(&attr, &mask);
            if (s != 0)
                errExitEN(s, "posix_spawnattr_setsigmask");
            attrp = &attr;
            break;
        }
    }
    /* Spawn the child. The name of the program to execute and the
       command-line arguments are taken from the command-line arguments
       of this program. The environment of the program execed in the
       child is made the same as the parent's environment. */
    s = posix_spawnp(&child_pid, argv[optind], file_actionsp, attrp,
                     &argv[optind], environ);
    if (s != 0)
        errExitEN(s, "posix_spawn");
    /* Destroy any objects that we created earlier */
    if (attrp != NULL) {
        s = posix_spawnattr_destroy(attrp);
        if (s != 0)
            errExitEN(s, "posix_spawnattr_destroy");
    }
    if (file_actionsp != NULL) {
        s = posix_spawn_file_actions_destroy(file_actionsp);
        if (s != 0)
            errExitEN(s, "posix_spawn_file_actions_destroy");
    }
    printf("PID of child: %jd\n", (intmax_t) child_pid);
    /* Monitor status of the child until it terminates */
    do {
```

```
            s = waitpid(child_pid, &status, WUNTRACED | WCONTINUED);
            if (s == -1)
                errExit("waitpid");
            printf("Child status: ");
            if (WIFEXITED(status)) {
                printf("exited, status=%d\n", WEXITSTATUS(status));
            } else if (WIFSIGNALED(status)) {
                printf("killed by signal %d\n", WTERMSIG(status));
            } else if (WIFSTOPPED(status)) {
                printf("stopped by signal %d\n", WSTOPSIG(status));
            } else if (WIFCONTINUED(status)) {
                printf("continued\n");
            }
        } while (!WIFEXITED(status) && !WIFSIGNALED(status));
        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

close(2), dup2(2), execl(2), execlp(2), fork(2), open(2), sched_setparam(2), sched_setscheduler(2), setpgid(2), setuid(2), sigaction(2), sigprocmask(2), posix_spawn_file_actions_addclose(3), posix_spawn_file_actions_adddup2(3), posix_spawn_file_actions_addopen(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), pthread_atfork(3), <spawn.h>, Base Definitions volume of POSIX.1-2001, http://www.opengroup.org/unix/online.html

COLOPHON