Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'posix_spawn_file_actions_addclose.3p' command

$ man posix_spawn_file_actions_addclose.3p

POSIX_SPAWN_FILE_ACTIONS_ADPOSIXEProgrammPOSIX_SPAWN_FILE_ACTIONS_ADDCLOSE(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawn_file_actions_addclose, posix_spawn_file_actions_addopen ? add close or open action to spawn file actions object (ADVANCED REAL? TIME)

SYNOPSIS

```
#include <spawn.h>
int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t
    *file_actions, int fildes);
int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
    *restrict file_actions, int fildes,
    const char *restrict path, int oflag, mode_t mode);
```

DESCRIPTION

These functions shall add or delete a close or open action to a spawn file actions object.

A spawn file actions object is of type posix_spawn_file_actions_t (de? fined in <spawn.h>) and is used to specify a series of actions to be performed by a posix_spawn() or posix_spawnp() operation in order to

arrive at the set of open file descriptors for the child process given the set of open file descriptors of the parent. POSIX.1?2008 does not define comparison or assignment operators for the type posix_spawn_file_actions_t.

A spawn file actions object, when passed to posix_spawn() or posix_spawnp(), shall specify how the set of open file descriptors in the calling process is transformed into a set of potentially open file descriptors for the spawned process. This transformation shall be as if the specified sequence of actions was performed exactly once, in the context of the spawned process (prior to execution of the new process image), in the order in which the actions were added to the object; ad? ditionally, when the new process image is executed, any file descriptor (from this new set) which has its FD_CLOEXEC flag set shall be closed (see posix_spawn()).

The posix_spawn_file_actions_addclose() function shall add a close ac? tion to the object referenced by file_actions that shall cause the file descriptor fildes to be closed (as if close(fildes) had been called) when a new process is spawned using this file actions object.

The posix_spawn_file_actions_addopen() function shall add an open ac? tion to the object referenced by file_actions that shall cause the file named by path to be opened (as if open(path, oflag, mode) had been called, and the returned file descriptor, if not fildes, had been changed to fildes) when a new process is spawned using this file ac? tions object. If fildes was already an open file descriptor, it shall be closed before the new file is opened.

The string described by path shall be copied by the posix_spawn_file_actions_addopen() function.

RETURN VALUE

Upon successful completion, these functions shall return zero; other? wise, an error number shall be returned to indicate the error.

ERRORS

The posix_spawn_file_actions_addopen() function shall fail if:

EBADF The value specified by fildes is negative or greater than or

equal to {OPEN_MAX}.

The posix_spawn_file_actions_addclose() function shall fail if:

EBADF The value specified by fildes is negative.

These functions may fail if:

EINVAL The value specified by file_actions is invalid.

ENOMEM Insufficient memory exists to add to the spawn file actions ob?

ject.

It shall not be considered an error for the fildes argument passed to these functions to specify a file descriptor for which the specified operation could not be performed at the time of the call. Any such er? ror will be detected when the associated file actions object is later used during a posix_spawn() or posix_spawnp() operation.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

Implementations may use file descriptors that must be inherited into child processes for the child process to remain conforming, such as for message catalog or tracing purposes. Therefore, an application that calls posix_spawn_file_actions_addclose() with an arbitrary integer risks non-conforming behavior, and this function can only portably be used to close file descriptor values that the application has obtained through explicit actions, or for the three file descriptors correspond? ing to the standard file streams. In order to avoid a race condition of leaking an unintended file descriptor into a child process, an applica? tion should consider opening all file descriptors with the FD_CLOEXEC bit set unless the file descriptor is intended to be inherited across exec.

RATIONALE

A spawn file actions object may be initialized to contain an ordered sequence of close(), dup2(), and open() operations to be used by

posix_spawn() or posix_spawnp() to arrive at the set of open file de? scriptors inherited by the spawned process from the set of open file descriptors in the parent at the time of the posix_spawn() or posix_spawnp() call. It had been suggested that the close() and dup2() operations alone are sufficient to rearrange file descriptors, and that files which need to be opened for use by the spawned process can be handled either by having the calling process open them before the posix_spawn() or posix_spawnp() call (and close them after), or by passing pathnames to the spawned process (in argv) so that it may open them itself. The standard developers recommend that applications use one of these two methods when practical, since detailed error status on a failed open operation is always available to the application this way. However, the standard developers feel that allowing a spawn file actions object to specify open operations is still appropriate because:

1. It is consistent with equivalent POSIX.5 (Ada) functionality.
2. It supports the I/O redirection paradigm commonly employed by POSIX programs designed to be invoked from a shell. When such a program is the child process, it may not be designed to open files on its own.
3. It allows file opens that might otherwise fail or violate file own? ership/access rights if executed by the parent process.

Regarding 2. above, note that the spawn open file action provides to posix_spawn() and posix_spawnp() the same capability that the shell re? direction operators provide to system(), only without the intervening execution of a shell; for example:

```
system ("myprog <file1 3<file2");
```

Regarding 3. above, note that if the calling process needs to open one or more files for access by the spawned process, but has insufficient spare file descriptors, then the open action is necessary to allow the open() to occur in the context of the child process after other file descriptors have been closed (that must remain open in the parent).

Additionally, if a parent is executed from a file having a ``set-user-id'' mode bit set and the POSIX_SPAWN_RESETIDS flag is set in the spawn

attributes, a file created within the parent process will (possibly in? correctly) have the parent's effective user ID as its owner, whereas a file created via an open() action during posix_spawn() or posix_spawnp() will have the parent's real ID as its owner; and an open by the parent process may successfully open a file to which the real user should not have access or fail to open a file to which the real user should have access.

File Descriptor Mapping

The standard developers had originally proposed using an array which specified the mapping of child file descriptors back to those of the parent. It was pointed out by the ballot group that it is not possible to reshuffle file descriptors arbitrarily in a library implementation of posix_spawn() or posix_spawnp() without provision for one or more spare file descriptor entries (which simply may not be available). Such an array requires that an implementation develop a complex strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor at the wrong time.

It was noted by a member of the Ada Language Bindings working group that the approved Ada Language Start_Process family of POSIX process primitives use a caller-specified set of file actions to alter the nor? mal fork()/exec semantics for inheritance of file descriptors in a very flexible way, yet no such problems exist because the burden of deter? mining how to achieve the final file descriptor mapping is completely on the application. Furthermore, although the file actions interface appears frightening at first glance, it is actually quite simple to im? plement in either a library or the kernel.

The posix_spawn_file_actions_addclose() function is not required to check whether the file descriptor is less than {OPEN_MAX} because on some implementations {OPEN_MAX} reflects the RLIMIT_NOFILE soft limit and therefore calling setrlimit() to reduce this limit can result in an {OPEN_MAX} value less than or equal to an already open file descriptor. Applications need to be able to close such file descriptors on spawn.

On implementations where {OPEN_MAX} does not change, it is recommended

that posix_spawn_file_actions_addclose() should return [EBADF] if fildes is greater than or equal to {OPEN_MAX}.

FUTURE DIRECTIONS

None.

SEE ALSO

close(), dup(), open(), posix_spawn(), posix_spawn_file_actions_ad? ddup2(), posix_spawn_file_actions_destroy()

The Base Definitions volume of POSIX.1?2017, <spawn.h>

COPYRIGHT