NAME

pidfile_open, **pidfile_write**, **pidfile_close**, **pidfile_remove** - library for PID files handling

LIBRARY

System Utilities Library (libutil, -lutil)

SYNOPSIS

#include <libutil.h>

*struct pidfh **
pidfile_open(*const char *path, mode_t mode, pid_t *pidptr*);

int
pidfile_write(*struct pidfh *pfh*);

int
pidfile_close(*struct pidfh *pfh*);

int
pidfile_remove(*struct pidfh *pfh*);

int
pidfile_fileno(*struct pidfh *pfh*);

DESCRIPTION

The **pidfile** family of functions allows daemons to handle PID files. It uses flopen(3) to lock a pidfile and detect already running daemons.

The **pidfile_open**() function opens (or creates) a file specified by the *path* argument and locks it. If *pidptr* argument is not NULL and file can not be locked, the function will use it to store a PID of an already running daemon or -1 in case daemon did not write its PID yet. The function does not write process' PID into the file here, so it can be used before **fork**()ing and exit with a proper error message when needed. If the *path* argument is NULL, */var/run/<progname>.pid* file will be used. The **pidfile_open**() function sets the O_CLOEXEC close-on-exec flag when opening the pidfile.

The **pidfile_write**() function writes process' PID into a previously opened file. The file is truncated before write, so calling the **pidfile_write**() function multiple times is supported.

The **pidfile_close**() function closes a pidfile. It should be used after daemon **fork**()s to start a child process.

The **pidfile_remove**() function closes and removes a pidfile.

The **pidfile_fileno**() function returns the file descriptor for the open pidfile.

RETURN VALUES

The **pidfile_open**() function returns a valid pointer to a *pidfh* structure on success, or NULL if an error occurs. If an error occurs, *errno* will be set.

The **pidfile_write**(), **pidfile_close**(), and **pidfile_remove**() functions return the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

The **pidfile_fileno**() function returns the low-level file descriptor. It returns -1 and sets *errno* if a NULL *pidfh* is specified, or if the pidfile is no longer open.

EXAMPLES

The following example shows in which order these functions should be used. Note that it is safe to pass NULL to **pidfile_write**(), **pidfile_remove**(), **pidfile_close**() and **pidfile_fileno**() functions.

```
struct pidfh *pfh;
pid_t otherpid, childpid;

pfh = pidfile_open("/var/run/daemon.pid", 0600, &otherpid);
if (pfh == NULL) {
        if (errno == EEXIST) {
                errx(EXIT_FAILURE, "Daemon already running, pid: %jd.",
                    (intmax_t)otherpid);
        }
        /* If we cannot create pidfile from other reasons, only warn. */
        warn("Cannot open or create pidfile");
        /*
         * Even though pfh is NULL we can continue, as the other pidfile_*
         * function can handle such situation by doing nothing except setting
         * errno to EDOOFUS.
         */
}

if (daemon(0, 0) == -1) {
        warn("Cannot daemonize");
        pidfile_remove(pfh);
        exit(EXIT_FAILURE);
```

```
}

pidfile_write(pfh);

for (;;) {
        /* Do work. */
        childpid = fork();
        switch (childpid) {
        case -1:
                syslog(LOG_ERR, "Cannot fork(): %s.", strerror(errno));
                break;
        case 0:
                pidfile_close(pfh);
                /* Do child work. */
                break;
        default:
                syslog(LOG_INFO, "Child %jd started.", (intmax_t)childpid);
                break;
        }
}

pidfile_remove(pfh);
exit(EXIT_SUCCESS);
```

ERRORS

The **pidfile_open**() function will fail if:

[EEXIST] Some process already holds the lock on the given pidfile, meaning that a daemon is already running. If *pidptr* argument is not NULL the function will use it to store a PID of an already running daemon or -1 in case daemon did not write its PID yet.

[ENAMETOOLONG] Specified pidfile’s name is too long.

[EINVAL] Some process already holds the lock on the given pidfile, but PID read from there is invalid.

The **pidfile_open**() function may also fail and set *errno* for any errors specified for the fstat(2), open(2), and read(2) calls.

The **pidfile_write**() function will fail if:

[EDOOFUS] Improper function use. Probably called before **pidfile_open**().

The **pidfile_write**() function may also fail and set *errno* for any errors specified for the fstat(2), ftruncate(2), and write(2) calls.

The **pidfile_close**() function may fail and set *errno* for any errors specified for the close(2) and fstat(2) calls.

The **pidfile_remove**() function will fail if:

[EDOOFUS] Improper function use. Probably called not from the process which made **pidfile_write**().

The **pidfile_remove**() function may also fail and set *errno* for any errors specified for the close(2), fstat(2), write(2), and unlink(2) system calls and the flopen(3) library function.

The **pidfile_fileno**() function will fail if:

[EDOOFUS] Improper function use. Probably called not from the process which used **pidfile_open**().

SEE ALSO

open(2), daemon(3), flopen(3)

HISTORY

The functions **pidfile_open**(), **pidfile_write**(), **pidfile_close**() and **pidfile_remove**() first appeared in FreeBSD 5.5.

AUTHORS

The **pidfile** functionality is based on ideas from John-Mark Gurney <*jmg@FreeBSD.org*>.

The code and manual page was written by Pawel Jakub Dawidek <*pjd@FreeBSD.org*>.