

**NAME**

**fileargs\_cinit**, **fileargs\_cinitnv**, **fileargs\_init**, **fileargs\_initnv**, **fileargs\_free**, **fileargs\_lstat**, **fileargs\_open**, **fileargs\_fopen** - library for handling files in capability mode

**LIBRARY**

library "libcap\_fileargs"

**SYNOPSIS**

```
#include <sys/nv.h>
```

```
#include <libcasper.h>
```

```
#include <casper/cap_fileargs.h>
```

*fileargs\_t* \*

```
fileargs_init(int argc, char *argv[], int flags, mode_t mode, cap_rights_t *rightsp, int operations);
```

*fileargs\_t* \*

```
fileargs_cinit(cap_channel_t *cas, int argc, char *argv[], int flags, mode_t mode, cap_rights_t *rightsp,  
int operations);
```

*fileargs\_t* \*

```
fileargs_cinitnv(cap_channel_t *cas, nvlist_t *limits);
```

*fileargs\_t* \*

```
fileargs_initnv(nvlist_t *limits);
```

*void*

```
fileargs_free(fileargs_t *fa);
```

*int*

```
fileargs_lstat(fileargs_t *fa, const char *path, struct stat *sb);
```

*int*

```
fileargs_open(fileargs_t *fa, const char *name);
```

*FILE* \*

```
fileargs_fopen(fileargs_t *fa, const char *name, const char *mode);
```

*char* \*

```
fileargs_realpath(fileargs_t *fa, const char *pathname, char *reserved_path);
```

**DESCRIPTION**

The library is used to simplify Capsicumizing a tools that are using file system. Idea behind the library is that we are passing a remaining *argc* and *argv* which contains a list of files that should be open for this program. The library will create a service that will serve those files.

The function **fileargs\_init()** create a service to the **system.fileargs**. The *argv* contains a list of files that should be opened. The argument can be set to NULL which will not create a service and all files will be prohibited to be opened. The *argc* argument contains a number of passed files. The *flags* argument limits opened files for either execution or reading and/or writing. The *mode* argument tells which what mode file should be created if the O\_CREATE flag is present . For more details of the *flags* and *mode* arguments see open(2). The *rightsp* argument contains a list of the capability rights which file should be limited to. For more details of the capability rights see cap\_rights\_init(3). The *operations* argument limits the operations that are available using **system.fileargs**. *operations* is a combination of:

FA\_OPEN

Allow **fileargs\_open()** and **fileargs\_fopen()**.

FA\_LSTAT

Allow **fileargs\_lstat()**.

FA\_REALPATH

Allow **fileargs\_realpath()**.

The function **fileargs\_cinit()** is equivalent to **fileargs\_init()** except that the connection to the Casper needs to be provided.

The functions **fileargs\_initnv()** and **fileargs\_cinitnv()** are respectively equivalent to **fileargs\_init()** and **fileargs\_cinit()** expect that all arguments all provided as nvlist(9). For details see *LIMITS*.

The *fileargs\_free* close connection to the **system.fileargs** service and free are structures. The function handle NULL argument.

The function **fileargs\_lstat()** is equivalent to lstat(2).

The functions **fileargs\_open()** and **fileargs\_fopen()** are respectively equivalent to open(2) and fopen(3) expect that all arguments are fetched from the *fileargs\_t* structure.

The function **fileargs\_realpath()** is equivalent to realpath(3).

**fileargs\_open()**, **fileargs\_lstat()**, **fileargs\_realpath()**, **fileargs\_cinitnv()**, **fileargs\_initnv()**, and

**fileargs\_fopen()** are reentrant but not thread-safe. That is, they may be called from separate threads only with different *cap\_channel\_t* arguments or with synchronization.

## LIMITS

This section describe which values and types should be used to pass arguments to the *system.fileargs* through the **fileargs\_initnv()** and **fileargs\_cinitnv()** functions. The *nvlist(9)* for that functions must contain the following values and types:

flags (NV\_TYPE\_NUMBER)

The *flags* limits opened files for either execution or reading and/or writing.

mode (NV\_TYPE\_NUMBER)

If in the *flags* argument the O\_CREATE flag was defined the *nvlist(9)* must contain the *mode*.

The *mode* argument tells which what mode file should be created.

operations (NV\_TYPE\_NUMBER)

The *operations* limits the usable operations for *system.fileargs*. The possible values are explained as *operations* argument with **fileargs\_init()**.

The *nvlist(9)* for that functions may contain the following values and types:

cap\_rights (NV\_TYPE\_BINARY)

The *cap\_rights* argument contains a list of the capability rights which file should be limited to.

(NV\_TYPE\_NULL)

Any number of NV\_TYPE\_NULL where the name of the element is name of the file which can be opened.

## EXAMPLES

The following example first parse some options and then create the **system.fileargs** service with remaining arguments.

```
int ch, fd, i;
cap_rights_t rights;
fileargs_t *fa;
```

```
while ((ch = getopt(argc, argv, "h")) != -1) {
    switch (ch) {
        case 'h':
        default:
```

```
        usage();
    }
}

argc -= optind;
argv += optind;

/* Create capability to the system.fileargs service. */
fa = fileargs_init(argc, argv, O_RDONLY, 0,
    cap_rights_init(&rights, CAP_READ), FA_OPEN);
if (fa == NULL)
    err(1, "unable to open system.fileargs service");

/* Enter capability mode sandbox. */
if (cap_enter() < 0 && errno != ENOSYS)
    err(1, "unable to enter capability mode");

/* Open files. */
for (i = 0; i < argc; i++) {
    fd = fileargs_open(fa, argv[i]);
    if (fd < 0)
        err(1, "unable to open file %s", argv[i]);
    printf("File %s opened in capability mode\n", argv[i]);
    close(fd);
}

fileargs_free(fa);
```

## SEE ALSO

cap\_enter(2), lstat(2), open(2), cap\_rights\_init(3), err(3), fopen(3), getopt(3), realpath(3), capsicum(4), nv(9)

## HISTORY

The **cap\_fileargs** service first appeared in FreeBSD 10.3.

## AUTHORS

Mariusz Zaborski <[oshogbo@FreeBSD.org](mailto:oshogbo@FreeBSD.org)>

## BUGS

The library "cap\_fileargs" included in FreeBSD is considered experimental, and should not be deployed

in production environments without careful consideration of the risks associated with the use of experimental operating system features.