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

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

$ man fopencookie.3

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

NAME

    fopencookie - opening a custom stream

SYNOPSIS

    #define _GNU_SOURCE   /* See feature_test_macros(7) */

    #include <stdio.h>

    FILE *fopencookie(void *cookie, const char *mode,

            cookie_io_functions_t io_funcs);

DESCRIPTION

    The fopencookie() function allows the programmer to create a custom im‐

    plementation for a standard I/O stream. This implementation can store

    the stream's data at a location of its own choosing; for example,

    fopencookie() is used to implement fmemopen(3), which provides a stream

    interface to data that is stored in a buffer in memory.

    In order to create a custom stream the programmer must:

    * Implement four "hook" functions that are used internally by the

      standard I/O library when performing I/O on the stream.

    * Define a "cookie" data type, a structure that provides bookkeeping

information (e.g., where to store data) used by the aforementioned hook functions. The standard I/O package knows nothing about the contents of this cookie (thus it is typed as void * when passed to fopencookie()), but automatically supplies the cookie as the first argument when calling the hook functions.

* Call fopencookie() to open a new stream and associate the cookie and hook functions with that stream.

The fopencookie() function serves a purpose similar to fopen(3): it opens a new stream and returns a pointer to a FILE object that is used to operate on that stream.

The cookie argument is a pointer to the caller's cookie structure that is to be associated with the new stream. This pointer is supplied as the first argument when the standard I/O library invokes any of the hook functions described below.

The mode argument serves the same purpose as for fopen(3). The follow‐ing modes are supported: r, w, a, r+, w+, and a+. See fopen(3) for de‐tails.

The io_funcs argument is a structure that contains four fields pointing to the programmer-defined hook functions that are used to implement this stream. The structure is defined as follows

```
typedef struct {
    cookie_read_function_t  *read;
    cookie_write_function_t *write;
    cookie_seek_function_t  *seek;
    cookie_close_function_t *close;
} cookie_io_functions_t;
```

The four fields are as follows:

cookie_read_function_t *read

This function implements read operations for the stream. When called, it receives three arguments:

```
ssize_t read(void *cookie, char *buf, size_t size);
```

The buf and size arguments are, respectively, a buffer into which input data can be placed and the size of that buffer. As

its function result, the read function should return the number
of bytes copied into buf, 0 on end of file, or -1 on error. The
read function should update the stream offset appropriately.
If *read is a null pointer, then reads from the custom stream
always return end of file.

cookie_write_function_t *write

This function implements write operations for the stream. When
called, it receives three arguments:

    ssize_t write(void *cookie, const char *buf, size_t size);

The buf and size arguments are, respectively, a buffer of data
to be output to the stream and the size of that buffer. As its
function result, the write function should return the number of
bytes copied from buf, or 0 on error. (The function must not
return a negative value.) The write function should update the
stream offset appropriately.

If *write is a null pointer, then output to the stream is dis?
carded.

cookie_seek_function_t *seek

This function implements seek operations on the stream.   When
called, it receives three arguments:

    int seek(void *cookie, off64_t *offset, int whence);

The *offset argument specifies the new file offset depending on
which of the following three values is supplied in whence:

SEEK_SET

    The stream offset should be set *offset bytes from the
    start of the stream.

SEEK_CUR

    *offset should be added to the current stream offset.

SEEK_END

    The stream offset should be set to the size of the stream
    plus *offset.

Before returning, the seek function should update *offset to in?
dicate the new stream offset.

As its function result, the seek function should return 0 on
success, and -1 on error.

If *seek is a null pointer, then it is not possible to perform
seek operations on the stream.

cookie_close_function_t *close

This function closes the stream. The hook function can do

things such as freeing buffers allocated for the stream. When

called, it receives one argument:

```
int close(void *cookie);
```

The cookie argument is the cookie that the programmer supplied

when calling fopencookie().

As its function result, the close function should return 0 on

success, and EOF on error.

If *close is NULL, then no special action is performed when the

stream is closed.

RETURN VALUE

On success fopencookie() returns a pointer to the new stream. On er?

ror, NULL is returned.

ATTRIBUTES

For an explanation of the terms used in this section, see at?

tributes(7).

??

?Interface   ? Attribute   ? Value   ?

???

?fopencookie() ? Thread safety ? MT-Safe ?

??

CONFORMING TO

This function is a nonstandard GNU extension.

EXAMPLES

The program below implements a custom stream whose functionality is

similar (but not identical) to that available via fmemopen(3). It im?

plements a stream whose data is stored in a memory buffer. The program

writes its command-line arguments to the stream, and then seeks through

the stream reading two out of every five characters and writing them to standard output. The following shell session demonstrates the use of the program:

```
$ ./a.out 'hello world'
/he/
/ w/
/d/
Reached end of file
```

Note that a more general version of the program below could be improved to more robustly handle various error situations (e.g., opening a stream with a cookie that already has an open stream; closing a stream that has already been closed).

Program source

```
#define _GNU_SOURCE
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define INIT_BUF_SIZE 4
struct memfile_cookie {
    char   *buf;       /* Dynamically sized buffer for data */
    size_t  allocated; /* Size of buf */
    size_t  endpos;    /* Number of characters in buf */
    off_t   offset;    /* Current file offset in buf */
};
ssize_t
memfile_write(void *c, const char *buf, size_t size)
{
    char *new_buff;
    struct memfile_cookie *cookie = c;
    /* Buffer too small? Keep doubling size until big enough */
    while (size + cookie->offset > cookie->allocated) {
```

```c
            new_buff = realloc(cookie->buf, cookie->allocated * 2);

            if (new_buff == NULL) {

                return -1;

            } else {

                cookie->allocated *= 2;

                cookie->buf = new_buff;

            }

        }

        memcpy(cookie->buf + cookie->offset, buf, size);

        cookie->offset += size;

        if (cookie->offset > cookie->endpos)

            cookie->endpos = cookie->offset;

        return size;

}

ssize_t

memfile_read(void *c, char *buf, size_t size)

{

        ssize_t xbytes;

        struct memfile_cookie *cookie = c;

        /* Fetch minimum of bytes requested and bytes available */

        xbytes = size;

        if (cookie->offset + size > cookie->endpos)

            xbytes = cookie->endpos - cookie->offset;

        if (xbytes < 0)     /* offset may be past endpos */

            xbytes = 0;

        memcpy(buf, cookie->buf + cookie->offset, xbytes);

        cookie->offset += xbytes;

        return xbytes;

}

int

memfile_seek(void *c, off64_t *offset, int whence)

{

        off64_t new_offset;
```

```c
    struct memfile_cookie *cookie = c;

    if (whence == SEEK_SET)

        new_offset = *offset;

    else if (whence == SEEK_END)

        new_offset = cookie->endpos + *offset;

    else if (whence == SEEK_CUR)

        new_offset = cookie->offset + *offset;

    else

        return -1;

    if (new_offset < 0)

        return -1;

    cookie->offset = new_offset;

    *offset = new_offset;

    return 0;

}
int
memfile_close(void *c)

{

    struct memfile_cookie *cookie = c;

    free(cookie->buf);

    cookie->allocated = 0;

    cookie->buf = NULL;

    return 0;

}
int
main(int argc, char *argv[])

{

    cookie_io_functions_t  memfile_func = {

        .read  = memfile_read,

        .write = memfile_write,

        .seek  = memfile_seek,

        .close = memfile_close

    };
```

```c
    FILE *stream;

    struct memfile_cookie mycookie;

    size_t nread;

    char buf[1000];

    /* Set up the cookie before calling fopencookie() */

    mycookie.buf = malloc(INIT_BUF_SIZE);

    if (mycookie.buf == NULL) {

        perror("malloc");

        exit(EXIT_FAILURE);

    }

    mycookie.allocated = INIT_BUF_SIZE;

    mycookie.offset = 0;

    mycookie.endpos = 0;

    stream = fopencookie(&mycookie,"w+", memfile_func);

    if (stream == NULL) {

        perror("fopencookie");

        exit(EXIT_FAILURE);

    }

    /* Write command-line arguments to our file */

    for (int j = 1; j < argc; j++)

        if (fputs(argv[j], stream) == EOF) {

            perror("fputs");

            exit(EXIT_FAILURE);

        }

    /* Read two bytes out of every five, until EOF */

    for (long p = 0; ; p += 5) {

        if (fseek(stream, p, SEEK_SET) == -1) {

            perror("fseek");

            exit(EXIT_FAILURE);

        }

        nread = fread(buf, 1, 2, stream);

        if (nread == 0) {

            if (ferror(stream) != 0) {
```

```
                fprintf(stderr, "fread failed\n");

                exit(EXIT_FAILURE);

            }

            printf("Reached end of file\n");

            break;

        }

        printf("/%.*s/\n", (int) nread, buf);

    }

    exit(EXIT_SUCCESS);

}
```

SEE ALSO

fclose(3), fmemopen(3), fopen(3), fseek(3)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A

description of the project, information about reporting bugs, and the

latest   version   of   this   page, can   be   found   at

https://www.kernel.org/doc/man-pages/.