NAME

fmemopen, open_memstream, open_wmemstream - open memory as stream

SYNOPSIS

#include <stdio.h>

FILE *fmemopen(void ***buf*, **size_t *size*, **const char ***mode*);

**FILE *open_memstream(char ****ptr*, **size_t ***sizeloc*);

#include <wchar.h>

**FILE *open_wmemstream(wchar_t ****ptr*, **size_t ***sizeloc*);

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

fmemopen(), **open_memstream**(), **open_wmemstream**():
Since glibc 2.10:
_XOPEN_SOURCE >= 700 || _POSIX_C_SOURCE >= 200809L
Before glibc 2.10:
_GNU_SOURCE

DESCRIPTION

The **fmemopen**() function opens a stream that permits the access specified by *mode*. The stream allows I/O to be performed on the string or memory buffer pointed to by *buf*. This buffer must be at least *size* bytes long.

The argument *mode* is the same as for fopen(3). If*mo de* specifies an append mode, then the initial file position is set to the location of the first null byte (0) in the buffer; otherwise the initial file position is set to the start of the buffer. Since glibc 2.9, the letter b may be specified as the second character in *mode*. This provides binary mode: writes don't implicitly add a terminating null byte, and fseek(3) **SEEK_END** is relative to the end of the buffer (i.e., the value specified by the *size* argument), rather than the current string length.

When a stream that has been opened for writing is flushed (fflush(3)) or closed (fclose(3)), a null byte is written at the end of the buffer if there is space. The caller should ensure that an extra byte is available in the buffer (and that *size* counts that byte) to allow for this.

Attempts to write more than *size* bytes to the buffer result in an error. (By default, such errors will be visible only when the *stdio* buffer is flushed. Disabling buffering with *setbuf(fp, NULL)* may be useful to detect errors at the time of an output operation. Alternatively, the caller can explicitly set *buf* as the stdio stream buffer, at the same time informing stdio of the buffer's size, using *setbuffer(fp, buf, size)*.)

In a stream opened for reading, null bytes (0) in the buffer do not cause read operations to return an end-of-file indication. A read from the buffer will only indicate end-of-file when the file pointer advances *size* bytes past the start of the buffer.

If *buf* is specified as NULL, then **fmemopen**() dynamically allocates a buffer *size* bytes long. This is useful for an application that wants to write data to a temporary buffer and then read it back again. The buffer is automatically freed when the stream is closed. Note that the caller has no way to obtain a pointer to the temporary buffer allocated by this call (but see **open_memstream**() below).

The **open_memstream**() function opens a stream for writing to a buffer. The buffer is dynamically allocated (as with malloc(3)), and automatically grows as required. After closing the stream, the caller should free(3) this buffer.

When the stream is closed (fclose(3)) or flushed (fflush(3)), the locations pointed to by *ptr* and *sizeloc* are updated to contain, respectively, a pointer to the buffer and the current size of the buffer. These values remain valid only as long as the caller performs no further output on the stream. If further output is performed, then the stream must again be flushed before trying to access these variables.

A null byte is maintained at the end of the buffer. This byte is *not* included in the size value stored at *sizeloc*.

The stream's file position can be changed with fseek(3) or fseeko(3). Moving the file position past the end of the data already written fills the intervening space with zeros.

The **open_wmemstream**() is similar to **open_memstream**(), but operates on wide characters instead of bytes.

RETURN VALUE

Upon successful completion **fmemopen**(), **open_memstream**() and **open_wmemstream**() return a *FILE* pointer. Otherwise, NULL is returned and*errno* is set to indicate the error.

VERSIONS

fmemopen() and **open_memstream**() were already available in glibc 1.0.x. **open_wmemstream**() is available since glibc 2.4.

CONFORMING TO

POSIX.1-2008. These functions are not specified in POSIX.1-2001, and are not widely available on other systems.

POSIX.1-2008 specifies that b in *mode* shall be ignored. However, Technical Corrigendum 1 adjusts the standard to allow implementation-specific treatment for this case, thus permitting the glibc treatment of b.

NOTES

There is no file descriptor associated with the file stream returned by these functions (i.e., fileno(3) will return an error if called on the returned stream).

BUGS

In glibc before version 2.7, seeking past the end of a stream created by **open_memstream**() does not enlarge the buffer; instead the fseek(3) call fails, returning -1.

If *size* is specified as zero, **fmemopen**() fails with the error **EINVAL**. It would be more consistent if this case successfully created a stream that then returned end of file on the first attempt at reading. Furthermore, POSIX.1-2008 does not specify a failure for this case.

Specifying append mode (a or a+) for **fmemopen**() sets the initial file position to the first null byte, but (if the file offset is reset to a location other than the end of the stream) does not force subsequent writes to append at the end of the stream.

If the *mode* argument to **fmemopen**() specifies append (a or a+), and the *size* argument does not cover a null byte in *buf*, then, according to POSIX.1-2008, the initial file position should be set to the next byte after the end of the buffer. However, in this case the glibc **fmemopen**() sets the file position to -1.

To specify binary mode for **fmemopen**() the b must be the *second* character in *mode*. Thus, for example, wb+ has the desired effect, but w+b does not. This is inconsistent with the treatment of *mode* by fopen(3).

The glibc 2.9 addition of binary mode for **fmemopen**() silently changed the ABI: previously, **fmemopen**() ignored b in *mode*.

EXAMPLE

The program below uses **fmemopen**() to open an input buffer, and **open_memstream**() to open a dynamically sized output buffer. The program scans its input string (taken from the program's first command-line argument) reading integers, and writes the squares of these integers to the output buffer. An example of the output produced by this program is the following:

```
$ ./a.out 1 23 43
size=11; ptr=1 529 1849
```

Program source

```
#define _GNU_SOURCE
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define handle_error(msg)
do { perror(msg); exit(EXIT_FAILURE); } while (0)

int
main(int argc, char *argv[])
{
FILE *out, *in;
int v, s;
size_t size;
char *ptr;

if (argc != 2) {
fprintf(stderr, Usage: %s <file>n, argv[0]);
exit(EXIT_FAILURE);
}

in = fmemopen(argv[1], strlen(argv[1]), r);
if (in == NULL)
handle_error(fmemopen);

out = open_memstream(&ptr, &size);
if (out == NULL)
handle_error(open_memstream);

for (;;) {
s = fscanf(in, %d, &v);
if (s <= 0)
break;

s = fprintf(out, %d , v * v);
if (s == -1)
handle_error(fprintf);
}
fclose(in);
fclose(out);
printf(size=%zu; ptr=%sn, size, ptr);
free(ptr);
exit(EXIT_SUCCESS);
}
```

SEE ALSO

fopen(3), fopencookie(3)

COLOPHON

This page is part of release 3.74 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 http://www.kernel.org/doc/man-pages/.