

NAME

eventfd - create a file descriptor for event notification

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <sys/eventfd.h>

int

eventfd(*unsigned int initval, int flags*);

int

eventfd_read(*int fd, eventfd_t *value*);

int

eventfd_write(*int fd, eventfd_t value*);

DESCRIPTION

eventfd() creates a special file descriptor with event counter or semaphore semantics, designed for interprocess communication. The returned file descriptor refers to a kernel object containing an unsigned 64-bit integer counter, which is initialized with the value of the *initval* argument.

The *flags* argument may contain the result of *or*'ing the following values:

EFD_CLOEXEC set FD_CLOEXEC on the file descriptor
EFD_NONBLOCK do not block on read/write operations
EFD_SEMAPHORE use semaphore semantics

File operations have the following semantics:

read(2) If the counter is zero, the call blocks until the counter becomes non-zero, unless EFD_NONBLOCK was set, in which case it would fail with EAGAIN instead.

If the counter is non-zero:

- If EFD_SEMAPHORE is not set, the current value of the counter is returned, and the value is reset to zero.
- If EFD_SEMAPHORE is set, the constant 1 is returned, and the value is

decremented by 1.

The numeric value is encoded as 64-bit (8 bytes) in host byte order. The `read(2)` call fails with `EINVAL` if there is less than 8 bytes available in the supplied buffer.

write(2) Adds the given value to the counter. The maximum value that can be stored in the counter is the maximum unsigned 64-bit integer value minus one (`0xfffffffffffffe`).

If the resulting value exceeds the maximum, the call would block until the value is reduced by `read(2)`, unless `EFD_NONBLOCK` was set, in which case it would fail with `EAGAIN` instead.

The numeric value is encoded as 64-bit (8 bytes) in host byte order. The `write(2)` call fails with `EINVAL` if there is less than 8 bytes available in the supplied buffer, or if the value `0xffffffffffff` is given.

poll(2) When receiving notifications via `poll(2)` / `ppoll(2)` / `select(2)` / `pselect(2)` / `kqueue(2)`, the following semantics apply:

- The file descriptor is readable when the counter is greater than zero.
- The file descriptor is writable when the counter is less than the maximum value.

File descriptors created by `eventfd()` are passable to other processes via `sendmsg(2)` and are preserved across `fork(2)`; in both cases the descriptors refer to the same counter from both processes. Unless `O_CLOEXEC` flag was specified, the created file descriptor will remain open across `execve(2)` system calls; see `close(2)`, `fcntl(2)` and `O_CLOEXEC` description.

eventfd_read() and **eventfd_write()** are thin wrappers around `read(2)` and `write(2)` system calls, provided for compatibility with glibc.

RETURN VALUES

If successful, **eventfd()** returns a non-negative integer, termed a file descriptor. It returns -1 on failure, and sets *errno* to indicate the error.

The **eventfd_read()** and **eventfd_write()** functions return 0 if the operation succeeded, -1 otherwise.

ERRORS

eventfd() may fail with:

- | | |
|----------|---|
| [EINVAL] | The <i>flags</i> argument given to eventfd() has unknown bits set. |
| [EMFILE] | The process has already reached its limit for open file descriptors. |
| [ENFILE] | The system file table is full. |
| [ENOMEM] | No memory was available to create the kernel object. |

SEE ALSO

close(2), kqueue(2), poll(2), read(2), select(2), write(2)

STANDARDS

The **eventfd()** system call is non-standard. It is present in Linux.

HISTORY

The **eventfd()** system call first appeared in FreeBSD 13.0.