****

**BAHIRDAR INSTITUTE OF TECHNOLOGY**

**OPERATING SYSTEM ASSIGNMENT**

**NAME: BEZAWIT ETSUBNEH**

**ID: BDU 1306205**

1.what/how/why this system calls?

2. Briefly describe about the list of parameters and flags

3. List the flags, their purpose with code implementation (give Example source code with output)

**#include <sys/eventfd.h>**

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

**eventfd**() creates an "eventfd object" that can be used as an

event wait/notify mechanism by user-space applications, and by

the kernel to notify user-space applications of events. The

object contains an unsigned 64-bit integer (*uint64\_t*) counter

that is maintained by the kernel. This counter is initialized

with the value specified in the argument *initval*.

As its return value, **eventfd**() returns a new file descriptor that

can be used to refer to the eventfd object.

The following values may be bitwise ORed in *flags* to change the

behavior of **eventfd**():

**EFD\_CLOEXEC** (since Linux 2.6.27)

Set the close-on-exec (**FD\_CLOEXEC**) flag on the new file

descriptor. See the description of the **O\_CLOEXEC** flag in

[open(2)](https://man7.org/linux/man-pages/man2/open.2.html) for reasons why this may be useful.

**EFD\_NONBLOCK** (since Linux 2.6.27)

Set the **O\_NONBLOCK** file status flag on the open file

description (see [open(2)](https://man7.org/linux/man-pages/man2/open.2.html)) referred to by the new file

descriptor. Using this flag saves extra calls to [fcntl(2)](https://man7.org/linux/man-pages/man2/fcntl.2.html)

to achieve the same result.

**EFD\_SEMAPHORE** (since Linux 2.6.30)

Provide semaphore-like semantics for reads from the new

file descriptor. See below.

In Linux up to version 2.6.26, the *flags* argument is unused, and

must be specified as zero.

The following operations can be performed on the file descriptor

returned by **eventfd**():

[read(2)](https://man7.org/linux/man-pages/man2/read.2.html)

Each successful [read(2)](https://man7.org/linux/man-pages/man2/read.2.html) returns an 8-byte integer. A

[read(2)](https://man7.org/linux/man-pages/man2/read.2.html) fails with the error **EINVAL** if the size of the

supplied buffer is less than 8 bytes.

The value returned by [read(2)](https://man7.org/linux/man-pages/man2/read.2.html) is in host byte order—that

is, the native byte order for integers on the host

machine.

The semantics of [read(2)](https://man7.org/linux/man-pages/man2/read.2.html) depend on whether the eventfd

counter currently has a nonzero value and whether the

**EFD\_SEMAPHORE** flag was specified when creating the eventfd

file descriptor:

\* If **EFD\_SEMAPHORE** was not specified and the eventfd

counter has a nonzero value, then a [read(2)](https://man7.org/linux/man-pages/man2/read.2.html) returns 8

bytes containing that value, and the counter's value is

reset to zero.

\* If **EFD\_SEMAPHORE** was specified and the eventfd counter

has a nonzero value, then a [read(2)](https://man7.org/linux/man-pages/man2/read.2.html) returns 8 bytes

containing the value 1, and the counter's value is

decremented by 1.

\* If the eventfd counter is zero at the time of the call

to [read(2)](https://man7.org/linux/man-pages/man2/read.2.html), then the call either blocks until the

counter becomes nonzero (at which time, the [read(2)](https://man7.org/linux/man-pages/man2/read.2.html)

proceeds as described above) or fails with the error

**EAGAIN** if the file descriptor has been made

nonblocking.

[write(2)](https://man7.org/linux/man-pages/man2/write.2.html)

A [write(2)](https://man7.org/linux/man-pages/man2/write.2.html) call adds the 8-byte integer value supplied in

its buffer to the counter. The maximum value that may be

stored in the counter is the largest unsigned 64-bit value

minus 1 (i.e., 0xfffffffffffffffe). If the addition would

cause the counter's value to exceed the maximum, then the

[write(2)](https://man7.org/linux/man-pages/man2/write.2.html) either blocks until a [read(2)](https://man7.org/linux/man-pages/man2/read.2.html) is performed on the

file descriptor, or fails with the error **EAGAIN** if the

file descriptor has been made nonblocking.

A [write(2)](https://man7.org/linux/man-pages/man2/write.2.html) fails with the error **EINVAL** if the size of the

supplied buffer is less than 8 bytes, or if an attempt is

made to write the value 0xffffffffffffffff.

[poll(2)](https://man7.org/linux/man-pages/man2/poll.2.html), [select(2)](https://man7.org/linux/man-pages/man2/select.2.html) (and similar)

The returned file descriptor supports [poll(2)](https://man7.org/linux/man-pages/man2/poll.2.html) (and

analogously [epoll(7)](https://man7.org/linux/man-pages/man7/epoll.7.html)) and [select(2)](https://man7.org/linux/man-pages/man2/select.2.html), as follows:

\* The file descriptor is readable (the [select(2)](https://man7.org/linux/man-pages/man2/select.2.html) *readfds*

argument; the [poll(2)](https://man7.org/linux/man-pages/man2/poll.2.html) **POLLIN** flag) if the counter has a

value greater than 0.

\* The file descriptor is writable (the [select(2)](https://man7.org/linux/man-pages/man2/select.2.html) *writefds*

argument; the [poll(2)](https://man7.org/linux/man-pages/man2/poll.2.html) **POLLOUT** flag) if it is possible

to write a value of at least "1" without blocking.

\* If an overflow of the counter value was detected, then

[select(2)](https://man7.org/linux/man-pages/man2/select.2.html) indicates the file descriptor as being both

readable and writable, and [poll(2)](https://man7.org/linux/man-pages/man2/poll.2.html) returns a **POLLERR**

event. As noted above, [write(2)](https://man7.org/linux/man-pages/man2/write.2.html) can never overflow the

counter. However an overflow can occur if 2^64 eventfd

"signal posts" were performed by the KAIO subsystem

(theoretically possible, but practically unlikely). If

an overflow has occurred, then [read(2)](https://man7.org/linux/man-pages/man2/read.2.html) will return that

maximum *uint64\_t* value (i.e., 0xffffffffffffffff).

The eventfd file descriptor also supports the other file-

descriptor multiplexing APIs: [pselect(2)](https://man7.org/linux/man-pages/man2/pselect.2.html) and [ppoll(2)](https://man7.org/linux/man-pages/man2/ppoll.2.html).

[close(2)](https://man7.org/linux/man-pages/man2/close.2.html)

When the file descriptor is no longer required it should

be closed. When all file descriptors associated with the

same eventfd object have been closed, the resources for

object are freed by the kernel.

A copy of the file descriptor created by **eventfd**() is inherited

by the child produced by [fork(2)](https://man7.org/linux/man-pages/man2/fork.2.html). The duplicate file descriptor

is associated with the same eventfd object. File descriptors

created by **eventfd**() are preserved across [execve(2)](https://man7.org/linux/man-pages/man2/execve.2.html), unless the

close-on-exec flag has been set.

**RETURN VALUE**[**top**](https://man7.org/linux/man-pages/man2/eventfd.2.html#top_of_page)

On success, **eventfd**() returns a new eventfd file descriptor. On

error, -1 is returned and [*errno*](https://man7.org/linux/man-pages/man3/errno.3.html) is set to indicate the error.

**ERRORS**[**top**](https://man7.org/linux/man-pages/man2/eventfd.2.html#top_of_page)

**EINVAL** An unsupported value was specified in *flags*.

**EMFILE** The per-process limit on the number of open file

descriptors has been reached.

**ENFILE** The system-wide limit on the total number of open files

has been reached.

**ENODEV** Could not mount (internal) anonymous inode device.

**ENOMEM** There was insufficient memory to create a new eventfd file

descriptor.

**VERSIONS**[**top**](https://man7.org/linux/man-pages/man2/eventfd.2.html#top_of_page)

**eventfd**() is available on Linux since kernel 2.6.22. Working

support is provided in glibc since version 2.8. The **eventfd2**()

system call (see NOTES) is available on Linux since kernel

2.6.27. Since version 2.9, the glibc **eventfd**() wrapper will

employ the **eventfd2**() system call, if it is supported by the

kernel.

**ATTRIBUTES**[**top**](https://man7.org/linux/man-pages/man2/eventfd.2.html#top_of_page)

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

[attributes(7)](https://man7.org/linux/man-pages/man7/attributes.7.html).

┌──────────────────────────────────────┬───────────────┬─────────┐

│**Interface** │ **Attribute** │ **Value** │

├──────────────────────────────────────┼───────────────┼─────────┤

│**eventfd**() │ Thread safety │ MT-Safe │

└──────────────────────────────────────┴───────────────┴─────────┘

**CONFORMING TO**[**top**](https://man7.org/linux/man-pages/man2/eventfd.2.html#top_of_page)

**eventfd**() and **eventfd2**() are Linux-specific.

**NOTES**[**top**](https://man7.org/linux/man-pages/man2/eventfd.2.html#top_of_page)

Applications can use an eventfd file descriptor instead of a pipe

(see [pipe(2)](https://man7.org/linux/man-pages/man2/pipe.2.html)) in all cases where a pipe is used simply to signal

events. The kernel overhead of an eventfd file descriptor is

much lower than that of a pipe, and only one file descriptor is

required (versus the two required for a pipe).

When used in the kernel, an eventfd file descriptor can provide a

bridge from kernel to user space, allowing, for example,

functionalities like KAIO (kernel AIO) to signal to a file

descriptor that some operation is complete.

A key point about an eventfd file descriptor is that it can be

monitored just like any other file descriptor using [select(2)](https://man7.org/linux/man-pages/man2/select.2.html),

[poll(2)](https://man7.org/linux/man-pages/man2/poll.2.html), or [epoll(7)](https://man7.org/linux/man-pages/man7/epoll.7.html). This means that an application can

simultaneously monitor the readiness of "traditional" files and

the readiness of other kernel mechanisms that support the eventfd

interface. (Without the **eventfd**() interface, these mechanisms

could not be multiplexed via [select(2)](https://man7.org/linux/man-pages/man2/select.2.html), [poll(2)](https://man7.org/linux/man-pages/man2/poll.2.html), or [epoll(7)](https://man7.org/linux/man-pages/man7/epoll.7.html).)

The current value of an eventfd counter can be viewed via the

entry for the corresponding file descriptor in the process's

*/proc/[pid]/fdinfo* directory. See [proc(5)](https://man7.org/linux/man-pages/man5/proc.5.html) for further details.

**C library/kernel differences**

There are two underlying Linux system calls: **eventfd**() and the

more recent **eventfd2**(). The former system call does not

implement a *flags* argument. The latter system call implements

the *flags* values described above. The glibc wrapper function

will use **eventfd2**() where it is available.

**Additional glibc features**

The GNU C library defines an additional type, and two functions

that attempt to abstract some of the details of reading and

writing on an eventfd file descriptor:

typedef uint64\_t eventfd\_t;

int eventfd\_read(int fd, eventfd\_t \*value);

int eventfd\_write(int fd, eventfd\_t value);

The functions perform the read and write operations on an eventfd

file descriptor, returning 0 if the correct number of bytes was

transferred, or -1 otherwise.

**EXAMPLES**[**top**](https://man7.org/linux/man-pages/man2/eventfd.2.html#top_of_page)

The following program creates an eventfd file descriptor and then

forks to create a child process. While the parent briefly

sleeps, the child writes each of the integers supplied in the

program's command-line arguments to the eventfd file descriptor.

When the parent has finished sleeping, it reads from the eventfd

file descriptor.

The following shell session shows a sample run of the program:

$ **./a.out 1 2 4 7 14**

Child writing 1 to efd

Child writing 2 to efd

Child writing 4 to efd

Child writing 7 to efd

Child writing 14 to efd

Child completed write loop

Parent about to read

Parent read 28 (0x1c) from efd

**Program source**

#include <sys/eventfd.h>

#include <unistd.h>

#include <inttypes.h> /\* Definition of PRIu64 & PRIx64 \*/

#include <stdlib.h>

#include <stdio.h>

#include <stdint.h> /\* Definition of uint64\_t \*/

#define handle\_error(msg) \

do { perror(msg); exit(EXIT\_FAILURE); } while (0)

int

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

{

int efd;

uint64\_t u;

ssize\_t s;

if (argc < 2) {

fprintf(stderr, "Usage: %s <num>...\n", argv[0]);

exit(EXIT\_FAILURE);

}

efd = eventfd(0, 0);

if (efd == -1)

handle\_error("eventfd");

switch (fork()) {

case 0:

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

printf("Child writing %s to efd\n", argv[j]);

u = strtoull(argv[j], NULL, 0);

/\* strtoull() allows various bases \*/

s = write(efd, &u, sizeof(uint64\_t));

if (s != sizeof(uint64\_t))

handle\_error("write");

}

printf("Child completed write loop\n");

exit(EXIT\_SUCCESS);

default:

sleep(2);

printf("Parent about to read\n");

s = read(efd, &u, sizeof(uint64\_t));

if (s != sizeof(uint64\_t))

handle\_error("read");

printf("Parent read %"PRIu64" (%#"PRIx64") from efd\n", u, u);

exit(EXIT\_SUCCESS);

case -1:

handle\_error("fork");

}

}

**REFERENCES**

**https://man7.org**