NAME

ioctl_userfaultfd – create a file descriptor for handling page faults in user space

SYNOPSIS

#include <sys/ioctl.h>

int ioctl(int *fd***, int** *cmd***, ...);**

DESCRIPTION

Various **ioctl**(2) operations can be performed on a userfaultfd object (created by a call to **userfaultfd**(2)) using calls of the form:

```
ioctl(fd, cmd, argp);
```

In the above, *fd* is a file descriptor referring to a userfaultfd object, *cmd* is one of the commands listed below, and *argp* is a pointer to a data structure that is specific to *cmd*.

The various **ioctl**(2) operations are described below. The **UFFDIO_API**, **UFFDIO_REGISTER**, and **UFFDIO_UNREGISTER** operations are used to *configure* userfaultfd behavior. These operations allow the caller to choose what features will be enabled and what kinds of events will be delivered to the application. The remaining operations are *range* operations. These operations enable the calling application to resolve page-fault events.

UFFDIO_API

(Since Linux 4.3.) Enable operation of the userfaultfd and perform API handshake.

The *argp* argument is a pointer to a *uffdio_api* structure, defined as:

```
struct uffdio_api {
    __u64 api;        /* Requested API version (input) */
    __u64 features;   /* Requested features (input/output) */
    __u64 ioctls;     /* Available ioctl() operations (output) */
};
```

The *api* field denotes the API version requested by the application.

The kernel verifies that it can support the requested API version, and sets the *features* and *ioctls* fields to bit masks representing all the available features and the generic **ioctl**(2) operations available.

For Linux kernel versions before 4.11, the *features* field must be initialized to zero before the call to *UFFDIO_API*, and zero (i.e., no feature bits) is placed in the *features* field by the kernel upon return from **ioctl**(2).

Starting from Linux 4.11, the *features* field can be used to ask whether particular features are supported and explicitly enable userfaultfd features that are disabled by default. The kernel always reports all the available features in the *features* field.

To enable userfaultfd features the application should set a bit corresponding to each feature it wants to enable in the *features* field. If the kernel supports all the requested features it will enable them. Otherwise it will zero out the returned *uffdio_api* structure and return **EINVAL**.

The following feature bits may be set:

UFFD_FEATURE_EVENT_FORK (since Linux 4.11)

When this feature is enabled, the userfaultfd objects associated with a parent process are duplicated into the child process during **fork**(2) and a **UFFD_EVENT_FORK** event is delivered to the userfaultfd monitor

UFFD_FEATURE_EVENT_REMAP (since Linux 4.11)

If this feature is enabled, when the faulting process invokes **mremap**(2), the userfaultfd monitor will receive an event of type **UFFD_EVENT_REMAP**.

UFFD_FEATURE_EVENT_REMOVE (since Linux 4.11)

If this feature is enabled, when the faulting process calls **madvise**(2) with the **MADV_DONT-NEED** or **MADV_REMOVE** advice value to free a virtual memory area the userfaultfd monitor will receive an event of type **UFFD_EVENT_REMOVE**.

UFFD_FEATURE_EVENT_UNMAP (since Linux 4.11)

> If this feature is enabled, when the faulting process unmaps virtual memory either explicitly with **munmap**(2), or implicitly during either **mmap**(2) or **mremap**(2). the userfaultfd monitor will receive an event of type **UFFD_EVENT_UNMAP**.

UFFD_FEATURE_MISSING_HUGETLBFS (since Linux 4.11)

> If this feature bit is set, the kernel supports registering userfaultfd ranges on hugetlbfs virtual memory areas

UFFD_FEATURE_MISSING_SHMEM (since Linux 4.11)

> If this feature bit is set, the kernel supports registering userfaultfd ranges on shared memory areas. This includes all kernel shared memory APIs: System V shared memory, **tmpfs**(5), shared mappings of */dev/zero*, **mmap**(2) with the **MAP_SHARED** flag set, **memfd_create**(2), and so on.

UFFD_FEATURE_SIGBUS (since Linux 4.14)

> If this feature bit is set, no page-fault events (**UFFD_EVENT_PAGEFAULT**) will be delivered. Instead, a **SIGBUS** signal will be sent to the faulting process. Applications using this feature will not require the use of a userfaultfd monitor for processing memory accesses to the regions registered with userfaultfd.

The returned *ioctls* field can contain the following bits:

1 << _UFFDIO_API

> The **UFFDIO_API** operation is supported.

1 << _UFFDIO_REGISTER

> The **UFFDIO_REGISTER** operation is supported.

1 << _UFFDIO_UNREGISTER

> The **UFFDIO_UNREGISTER** operation is supported.

This **ioctl**(2) operation returns 0 on success. On error, −1 is returned and *errno* is set to indicate the cause of the error. Possible errors include:

EFAULT

> *argp* refers to an address that is outside the calling process's accessible address space.

EINVAL

> The userfaultfd has already been enabled by a previous **UFFDIO_API** operation.

EINVAL

> The API version requested in the *api* field is not supported by this kernel, or the *features* field passed to the kernel includes feature bits that are not supported by the current kernel version.

UFFDIO_REGISTER

(Since Linux 4.3.) Register a memory address range with the userfaultfd object. The pages in the range must be "compatible".

Up to Linux kernel 4.11, only private anonymous ranges are compatible for registering with **UFFDIO_REGISTER**.

Since Linux 4.11, hugetlbfs and shared memory ranges are also compatible with **UFFDIO_REGISTER**.

The *argp* argument is a pointer to a *uffdio_register* structure, defined as:

```
struct uffdio_range {
    __u64 start;    /* Start of range */
    __u64 len;      /* Length of range (bytes) */
};

struct uffdio_register {
    struct uffdio_range range;
    __u64 mode;     /* Desired mode of operation (input) */
    __u64 ioctls;   /* Available ioctl() operations (output) */
```

```
    };
```

The *range* field defines a memory range starting at *start* and continuing for *len* bytes that should be handled by the userfaultfd.

The *mode* field defines the mode of operation desired for this memory region. The following values may be bitwise ORed to set the userfaultfd mode for the specified range:

UFFDIO_REGISTER_MODE_MISSING
>   Track page faults on missing pages.

UFFDIO_REGISTER_MODE_WP
>   Track page faults on write-protected pages.

Currently, the only supported mode is **UFFDIO_REGISTER_MODE_MISSING**.

If the operation is successful, the kernel modifies the *ioctls* bit-mask field to indicate which **ioctl**(2) operations are available for the specified range. This returned bit mask is as for **UFFDIO_API**.

This **ioctl**(2) operation returns 0 on success. On error, −1 is returned and *errno* is set to indicate the cause of the error. Possible errors include:

EBUSY
>   A mapping in the specified range is registered with another userfaultfd object.

EFAULT
>   *argp* refers to an address that is outside the calling process's accessible address space.

EINVAL
>   An invalid or unsupported bit was specified in the *mode* field; or the *mode* field was zero.

EINVAL
>   There is no mapping in the specified address range.

EINVAL
>   *range.start* or *range.len* is not a multiple of the system page size; or, *range.len* is zero; or these fields are otherwise invalid.

EINVAL
>   There as an incompatible mapping in the specified address range.

UFFDIO_UNREGISTER
>   (Since Linux 4.3.) Unregister a memory address range from userfaultfd. The pages in the range must be "compatible" (see the description of **UFFDIO_REGISTER**.)

>   The address range to unregister is specified in the *uffdio_range* structure pointed to by *argp*.

>   This **ioctl**(2) operation returns 0 on success. On error, −1 is returned and *errno* is set to indicate the cause of the error. Possible errors include:

EINVAL
>   Either the *start* or the *len* field of the *ufdio_range* structure was not a multiple of the system page size; or the *len* field was zero; or these fields were otherwise invalid.

EINVAL
>   There as an incompatible mapping in the specified address range.

EINVAL
>   There was no mapping in the specified address range.

UFFDIO_COPY
>   (Since Linux 4.3.) Atomically copy a continuous memory chunk into the userfault registered range and optionally wake up the blocked thread. The source and destination addresses and the number of bytes to copy are specified by the *src*, *dst*, and *len* fields of the *uffdio_copy* structure pointed to by *argp*:

```
        struct uffdio_copy {
            __u64 dst;    /* Destination of copy */
```

```
        __u64 src;    /* Source of copy */
        __u64 len;    /* Number of bytes to copy */
        __u64 mode;   /* Flags controlling behavior of copy */
        __s64 copy;   /* Number of bytes copied, or negated error */
    };
```

The following value may be bitwise ORed in *mode* to change the behavior of the **UFFDIO_COPY** operation:

UFFDIO_COPY_MODE_DONTWAKE
> Do not wake up the thread that waits for page-fault resolution

The *copy* field is used by the kernel to return the number of bytes that was actually copied, or an error (a negated *errno*-style value). If the value returned in *copy* doesn't match the value that was specified in *len*, the operation fails with the error **EAGAIN**. The *copy* field is output-only; it is not read by the **UFFDIO_COPY** operation.

This **ioctl**(2) operation returns 0 on success. In this case, the entire area was copied. On error, −1 is returned and *errno* is set to indicate the cause of the error. Possible errors include:

EAGAIN
> The number of bytes copied (i.e., the value returned in the *copy* field) does not equal the value that was specified in the *len* field.

EINVAL
> Either *dst* or *len* was not a multiple of the system page size, or the range specified by *src* and *len* or *dst* and *len* was invalid.

EINVAL
> An invalid bit was specified in the *mode* field.

ENOENT (since Linux 4.11)
> The faulting process has changed its virtual memory layout simultaneously with an outstanding *UFFDIO_COPY* operation.

ENOSPC (from Linux 4.11 until Linux 4.13)
> The faulting process has exited at the time of a *UFFDIO_COPY* operation.

ESRCH (since Linux 4.13)
> The faulting process has exited at the time of a *UFFDIO_COPY* operation.

UFFDIO_ZEROPAGE
(Since Linux 4.3.) Zero out a memory range registered with userfaultfd.

The requested range is specified by the *range* field of the *uffdio_zeropage* structure pointed to by *argp*:

```
    struct uffdio_zeropage {
        struct uffdio_range range;
        __u64 mode;     /* Flags controlling behavior of copy */
        __s64 zeropage; /* Number of bytes zeroed, or negated error */
    };
```

The following value may be bitwise ORed in *mode* to change the behavior of the **UFFDIO_ZEROPAGE** operation:

UFFDIO_ZEROPAGE_MODE_DONTWAKE
> Do not wake up the thread that waits for page-fault resolution.

The *zeropage* field is used by the kernel to return the number of bytes that was actually zeroed, or an error in the same manner as **UFFDIO_COPY**. If the value returned in the *zeropage* field doesn't match the value that was specified in *range.len*, the operation fails with the error **EAGAIN**. The *zeropage* field is output-only; it is not read by the **UFFDIO_ZEROPAGE** operation.

This **ioctl**(2) operation returns 0 on success. In this case, the entire area was zeroed. On error, −1 is returned and *errno* is set to indicate the cause of the error. Possible errors include:

EAGAIN
> The number of bytes zeroed (i.e., the value returned in the *zeropage* field) does not equal the value that was specified in the *range.len* field.

EINVAL
> Either *range.start* or *range.len* was not a multiple of the system page size; or *range.len* was zero; or the range specified was invalid.

EINVAL
> An invalid bit was specified in the *mode* field.

ESRCH (since Linux 4.13)
> The faulting process has exited at the time of a *UFFDIO_ZEROPAGE* operation.

UFFDIO_WAKE
> (Since Linux 4.3.) Wake up the thread waiting for page-fault resolution on a specified memory address range.
>
> The **UFFDIO_WAKE** operation is used in conjunction with **UFFDIO_COPY** and **UFFDIO_ZE-ROPAGE** operations that have the **UFFDIO_COPY_MODE_DONTWAKE** or **UFFDIO_ZE-ROPAGE_MODE_DONTWAKE** bit set in the *mode* field. The userfault monitor can perform several **UFFDIO_COPY** and **UFFDIO_ZEROPAGE** operations in a batch and then explicitly wake up the faulting thread using **UFFDIO_WAKE**.
>
> The *argp* argument is a pointer to a *uffdio_range* structure (shown above) that specifies the address range.
>
> This **ioctl**(2) operation returns 0 on success. On error, −1 is returned and *errno* is set to indicate the cause of the error. Possible errors include:
>
> **EINVAL**
> > The *start* or the *len* field of the *ufdio_range* structure was not a multiple of the system page size; or *len* was zero; or the specified range was otherwise invalid.

RETURN VALUE
> See descriptions of the individual operations, above.

ERRORS
> See descriptions of the individual operations, above. In addition, the following general errors can occur for all of the operations described above:
>
> **EFAULT**
> > *argp* does not point to a valid memory address.
>
> **EINVAL**
> > (For all operations except **UFFDIO_API**.) The userfaultfd object has not yet been enabled (via the **UFFDIO_API** operation).

CONFORMING TO
> These **ioctl**(2) operations are Linux-specific.

BUGS
> In order to detect available userfault features and enable some subset of those features the userfaultfd file descriptor must be closed after the first **UFFDIO_API** operation that queries features availability and re-opened before the second **UFFDIO_API** operation that actually enables the desired features.

EXAMPLE
> See **userfaultfd**(2).

SEE ALSO
> **ioctl**(2), **mmap**(2), **userfaultfd**(2)
>
> *Documentation/admin-guide/mm/userfaultfd.rst* in the Linux kernel source tree

COLOPHON

This page is part of release 5.05 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/.