NAME

fuse − format and options for the fuse file systems

DESCRIPTION

FUSE (Filesystem in Userspace) is a simple interface for userspace programs to export a virtual filesystem to the Linux kernel. FUSE also aims to provide a secure method for non privileged users to create and mount their own filesystem implementations.

CONFIGURATION

Some options regarding mount policy can be set in the file */etc/fuse.conf*. Currently these options are:

mount_max = NNN
Set the maximum number of FUSE mounts allowed to non-root users. The default is 1000.

user_allow_other
Allow non-root users to specify the **allow_other** or **allow_root** mount options (see below).

OPTIONS

Most of the generic mount options described in **mount** are supported (**ro**, **rw**, **suid**, **nosuid**, **dev**, **nodev**, **exec**, **noexec**, **atime**, **noatime**, **sync**, **async**, **dirsync**). Filesystems are mounted with **nodev,nosuid** by default, which can only be overridden by a privileged user.

General mount options:

These are FUSE specific mount options that can be specified for all filesystems:

default_permissions
By default FUSE doesn't check file access permissions, the filesystem is free to implement it's access policy or leave it to the underlying file access mechanism (e.g. in case of network filesystems). This option enables permission checking, restricting access based on file mode. This is option is usually useful together with the **allow_other** mount option.

allow_other
This option overrides the security measure restricting file access to the user mounting the filesystem. So all users (including root) can access the files. This option is by default only allowed to root, but this restriction can be removed with a configuration option described in the previous section.

allow_root
This option is similar to **allow_other** but file access is limited to the user mounting the filesystem and root. This option and **allow_other** are mutually exclusive.

kernel_cache
This option disables flushing the cache of the file contents on every **open**(2). This should only be enabled on filesystems, where the file data is never changed externally (not through the mounted FUSE filesystem). Thus it is not suitable for network filesystems and other *intermediate* filesystems.

NOTE: if this option is not specified (and neither **direct_io**) data is still cached after the **open**(2), so a **read**(2) system call will not always initiate a read operation.

auto_cache
This option enables automatic flushing of the data cache on **open**(2). The cache will only be flushed if the modification time or the size of the file has changed.

large_read
Issue large read requests. This can improve performance for some filesystems, but can also degrade performance. This option is only useful on 2.4.X kernels, as on 2.6 kernels requests size is automatically determined for optimum performance.

direct_io
This option disables the use of page cache (file content cache) in the kernel for this filesystem. This has several affects:

1. Each **read**(2) or **write**(2) system call will initiate one or more read or write operations, data will not be cached in the kernel.

2. The return value of the read() and write() system calls will correspond to the return values of the read and write operations. This is useful for example if the file size is not known in advance (before reading it).

max_read=N
With this option the maximum size of read operations can be set. The default is infinite. Note that the size of read requests is limited anyway to 32 pages (which is 128kbyte on i386).

max_readahead=N
Set the maximum number of bytes to read-ahead. The default is determined by the kernel. On linux-2.6.22 or earlier it's 131072 (128kbytes)

max_write=N
Set the maximum number of bytes in a single write operation. The default is 128kbytes. Note, that due to various limitations, the size of write requests can be much smaller (4kbytes). This limitation will be removed in the future.

async_read
Perform reads asynchronously. This is the default

sync_read
Perform all reads (even read-ahead) synchronously.

hard_remove
The default behavior is that if an open file is deleted, the file is renamed to a hidden file (**.fuse_hiddenXXX**), and only removed when the file is finally released. This relieves the filesystem implementation of having to deal with this problem. This option disables the hiding behavior, and files are removed immediately in an unlink operation (or in a rename operation which overwrites an existing file).

It is recommended that you not use the hard_remove option. When hard_remove is set, the following libc functions fail on unlinked files (returning errno of **ENOENT**): **read**(2), **write**(2), **fsync**(2), **close**(2), **f*xattr**(2), **ftruncate**(2), **fstat**(2), **fchmod**(2), **fchown**(2)

debug Turns on debug information printing by the library.

fsname=NAME
Sets the filesystem source (first field in */etc/mtab*). The default is the mount program name.

subtype=TYPE
Sets the filesystem type (third field in */etc/mtab*). The default is the mount program name. If the kernel suppports it, */etc/mtab* and */proc/mounts* will show the filesystem type as **fuse.TYPE**

If the kernel doesn't support subtypes, the source filed will be **TYPE#NAME**, or if **fsname** option is not specified, just **TYPE**.

use_ino
Honor the *st_ino* field in kernel functions **getattr()** and **fill_dir()**. This value is used to fill in the *st_ino* field in the **stat**(2), **lstat**(2), **fstat**(2) functions and the *d_ino* field in the **readdir**(2) function. The filesystem does not have to guarantee uniqueness, however some applications rely on this value being unique for the whole filesystem.

readdir_ino
If **use_ino** option is not given, still try to fill in the *d_ino* field in **readdir**(2). If the name was previously looked up, and is still in the cache, the inode number found there will be used. Otherwise it will be set to **-1**. If **use_ino** option is given, this option is ignored.

nonempty
Allows mounts over a non-empty file or directory. By default these mounts are rejected to prevent accidental covering up of data, which could for example prevent automatic backup.

umask=M
Override the permission bits in *st_mode* set by the filesystem. The resulting permission bits are the ones missing from the given umask value. The value is given in octal representation.

uid=N Override the *st_uid* field set by the filesystem (N is numeric).

gid=N Override the *st_gid* field set by the filesystem (N is numeric).

blkdev Mount a filesystem backed by a block device. This is a privileged option. The device must be specified with the **fsname=NAME** option.

entry_timeout=T
The timeout in seconds for which name lookups will be cached. The default is 1.0 second. For all the timeout options, it is possible to give fractions of a second as well (e.g. **entry_timeout=2.8**)

negative_timeout=T
The timeout in seconds for which a negative lookup will be cached. This means, that if file did not exist (lookup retuned **ENOENT**), the lookup will only be redone after the timeout, and the file/directory will be assumed to not exist until then. The default is 0.0 second, meaning that caching negative lookups are disabled.

attr_timeout=T
The timeout in seconds for which file/directory attributes are cached. The default is 1.0 second.

ac_attr_timeout=T
The timeout in seconds for which file attributes are cached for the purpose of checking if **auto_cache** should flush the file data on open. The default is the value of **attr_timeout**

intr Allow requests to be interrupted. Turning on this option may result in unexpected behavior, if the filesystem does not support request interruption.

intr_signal=NUM
Specify which signal number to send to the filesystem when a request is interrupted. The default is hardcoded to USR1.

modules=M1[:M2...]
Add modules to the filesystem stack. Modules are pushed in the order they are specified, with the original filesystem being on the bottom of the stack.

FUSE MODULES (STACKING)

Modules are filesystem stacking support to high level API. Filesystem modules can be built into libfuse or loaded from shared object

iconv
Perform file name character set conversion. Options are:

from_code=CHARSET
Character set to convert from (see **iconv -l** for a list of possible values). Default is **UTF-8**.

to_code=CHARSET
Character set to convert to. Default is determined by the current locale.

subdir
Prepend a given directory to each path. Options are:

subdir=DIR
Directory to prepend to all paths. This option is *mandatory*.

rellinks
Transform absolute symlinks into relative

norellinks
Do not transform absolute symlinks into relative. This is the default.

SECURITY

The fusermount program is installed set-user-gid to fuse. This is done to allow users from fuse group to mount their own filesystem implementations. There must however be some limitations, in order to prevent Bad User from doing nasty things. Currently those limitations are:

1. The user can only mount on a mountpoint, for which it has write permission
2. The mountpoint is not a sticky directory which isn't owned by the user (like */tmp* usually is)
3. No other user (including root) can access the contents of the mounted filesystem.

NOTE

FUSE filesystems are unmounted using the **fusermount**(1) command (**fusermount -u mountpoint**).

AUTHORS

The main author of FUSE is Miklos Szeredi <mszeredi@inf.bme.hu>.

This man page was written by Bastien Roucaries <roucaries.bastien+debian@gmail.com> for the Debian GNU/Linux distribution (but it may be used by others) from README file.

SEE ALSO

fusermount(1) mount(8)