## NAME

virtiofsd − QEMU virtio-fs shared file system daemon

## SYNOPSIS

**virtiofsd** [*OPTIONS*]

## DESCRIPTION

Share a host directory tree with a guest through a virtio−fs device. This program is a vhost−user backend that implements the virtio−fs device. Each virtio−fs device instance requires its own virtiofsd process.

This program is designed to work with QEMU's **−−device vhost−user−fs−pci** but should work with any virtual machine monitor (VMM) that supports vhost−user. See the Examples section below.

This program must be run as the root user. The program drops privileges where possible during startup although it must be able to create and access files with any uid/gid:

- The ability to invoke syscalls is limited using seccomp(2).
- Linux capabilities(7) are dropped.

In "namespace" sandbox mode the program switches into a new file system namespace and invokes pivot_root(2) to make the shared directory tree its root. A new pid and net namespace is also created to isolate the process.

In "chroot" sandbox mode the program invokes chroot(2) to make the shared directory tree its root. This mode is intended for container environments where the container runtime has already set up the namespaces and the program does not have permission to create namespaces itself.

Both sandbox modes prevent "file system escapes" due to symlinks and other file system objects that might lead to files outside the shared directory.

## OPTIONS

**−h, −−help**
Print help.

**−V, −−version**
Print version.

**−d** Enable debug output.

**−−syslog**
Print log messages to syslog instead of stderr.

**−o OPTION**

- debug − Enable debug output.
- flock|no_flock − Enable/disable flock. The default is **no_flock**.
- modcaps=CAPLIST Modify the list of capabilities allowed; CAPLIST is a colon separated list of capabilities, each preceded by either + or −, e.g. "+sys_admin:−chown".
- log_level=LEVEL − Print only log messages matching LEVEL or more severe. LEVEL is one of **err**, **warn**, **info**, or **debug**. The default is **info**.
- posix_lock|no_posix_lock − Enable/disable remote POSIX locks. The default is **no_posix_lock**.
- readdirplus|no_readdirplus − Enable/disable readdirplus. The default is **readdirplus**.
- sandbox=namespace|chroot − Sandbox mode: − namespace: Create mount, pid, and net namespaces and pivot_root(2) into the shared directory. − chroot: chroot(2) into shared directory (use in containers). The default is "namespace".

- source=PATH − Share host directory tree located at PATH. This option is required.
- timeout=TIMEOUT − I/O timeout in seconds. The default depends on cache= option.
- writeback|no_writeback − Enable/disable writeback cache. The cache allows the FUSE client to buffer and merge write requests. The default is **no_writeback**.
- xattr|no_xattr − Enable/disable extended attributes (xattr) on files and directories. The default is **no_xattr**.
- posix_acl|no_posix_acl − Enable/disable posix acl support. Posix ACLs are disabled by default.

**−−socket−path=PATH**
Listen on vhost−user UNIX domain socket at PATH.

**−−socket−group=GROUP**
Set the vhost−user UNIX domain socket gid to GROUP.

**−−fd=FDNUM**
Accept connections from vhost−user UNIX domain socket file descriptor FDNUM. The file descriptor must already be listening for connections.

**−−thread−pool−size=NUM**
Restrict the number of worker threads per request queue to NUM. The default is 64.

**−−cache=none|auto|always**
Select the desired trade−off between coherency and performance. **none** forbids the FUSE client from caching to achieve best coherency at the cost of performance. **auto** acts similar to NFS with a 1 second metadata cache timeout. **always** sets a long cache lifetime at the e xpense of coherency. The default is **auto**.

## EXTENDED ATTRIBUTE (XATTR) MAPPING

By default the name of xattr's used by the client are passed through to the server file system. This can be a problem where either those xattr names are used by something on the server (e.g. selinux client/server confusion) or if the **virtiofsd** is running in a container with restricted privileges where it cannot access some attributes.

### Mapping syntax

A mapping of xattr names can be made using −o xattrmap=mapping where the **mapping** string consists of a series of rules.

The first matching rule terminates the mapping. The set of rules must include a terminating rule to match any remaining attributes at the end.

Each rule consists of a number of fields separated with a separator that is the first non−white space character in the rule. This separator must then be used for the whole rule. White space may be added before and after each rule.

Using ':' as the separator a rule is of the form:

**:type:scope:key:prepend:**

**scope** is:

- **'client' − match 'key' against a xattr name from the client for**
  setxattr/getxattr/removexattr
-

**'server' − match 'prepend' against a xattr name from the server**
for listxattr

- **'all' − can be used to make a single rule where both the server**
and client matches are triggered.

**type** is one of:

- 'prefix' − is designed to prepend and strip a prefix; the modified attributes then being passed on to the client/server.
- 'ok' − Causes the rule set to be terminated when a match is found while allowing matching xattr's through unchanged. It is intended both as a way of explicitly terminating the list of rules, and to allow some xattr's to skip following rules.
- 'bad' − If a client tries to use a name matching 'key' it's denied using EPERM; when the server passes an attribute name matching 'prepend' it's hidden. In many ways it's use is very like 'ok' as either an explicit terminator or for special handling of certain patterns.
- 'unsupported' − If a client tries to use a name matching 'key' it's denied using ENOTSUP; when the server passes an attribute name matching 'prepend' it's hidden. In many ways it's use is very like 'ok' as either an explicit terminator or for special handling of certain patterns.

**key** is a string tested as a prefix on an attribute name originating on the client. It maybe empty in which case a 'client' rule will always match on client names.

**prepend** is a string tested as a prefix on an attribute name originating on the server, and used as a new prefix. It may be empty in which case a 'server' rule will always match on all names from the server.

e.g.:

**:prefix:client:trusted.:user.virtiofs.:**

will match 'trusted.' attributes in client calls and prefix them before passing them to the server.

**:prefix:server::user.virtiofs.:**

will strip 'user.virtiofs.' from all server replies.

**:prefix:all:trusted.:user.virtiofs.:**

combines the previous two cases into a single rule.

**:ok:client:user.::**

will allow get/set xattr for 'user.' xattr's and ignore following rules.

**:ok:server::security.:**

will pass 'securty.' xattr's in listxattr from the server and ignore following rules.

**:ok:all:::**

will terminate the rule search passing any remaining attributes in both directions.

**:bad:server::security.:**

would hide 'security.' xattr's in listxattr from the server.

A simpler 'map' type provides a shorter syntax for the common case:

**:map:key:prepend:**

The 'map' type adds a number of separate rules to add **prepend** as a prefix to the matched **key** (or all attributes if **key** is empty). There may be at most one 'map' rule and it must be the last rule in the set.

Note: When the 'security.capability' xattr is remapped, the daemon has to do extra work to remove it during many operations, which the host kernel normally does itself.

**Security considerations**

Operating systems typically partition the xattr namespace using well defined name prefixes. Each partition may have different access controls applied. For example, on Linux there are multiple partitions

- **system.*** − access varies depending on attribute & filesystem
- **security.*** − only processes with CAP_SYS_ADMIN
- **trusted.*** − only processes with CAP_SYS_ADMIN
- **user.*** − any process granted by file permissions / ownership

While other OS such as FreeBSD have different name prefixes and access control rules.

When remapping attributes on the host, it is important to ensure that the remapping does not allow a guest user to evade the guest access control rules.

Consider if **trusted.*** from the guest was remapped to **user.virtiofs.trusted*** in the host. An unprivileged user in a Linux guest has the ability to write to xattrs under **user.***. Thus the user can evade the access control restriction on **trusted.*** by instead writing to **user.virtiofs.trusted.***.

As noted above, the partitions used and access controls applied, will vary across guest OS, so it is not wise to try to predict what the guest OS will use.

The simplest way to avoid an insecure configuration is to remap all xattrs at once, to a given fixed prefix. This is shown in example (1) below.

If selectively mapping only a subset of xattr prefixes, then rules must be added to explicitly block direct access to the target of the remapping. This is shown in example (2) below.

**Mapping examples**

1. Prefix all attributes with 'user.virtiofs.'

```
-o xattrmap=":prefix:all::user.virtiofs.::bad:all:::"
```

This uses two rules, using : as the field separator; the first rule prefixes and strips 'user.virtiofs.', the second rule hides any non−prefixed attributes that the host set.

This is equivalent to the 'map' rule:

```
-o xattrmap=":map::user.virtiofs.:"
```

2. Prefix 'trusted.' attributes, allow others through

```
"/prefix/all/trusted./user.virtiofs./
 /bad/server//trusted./
 /bad/client/user.virtiofs.//
```

```
 /ok/all///"
```

Here there are four rules, using / as the field separator, and also demonstrating that new lines can be included between rules. The first rule is the prefixing of 'trusted.' and stripping of 'user.virtiofs.'. The second rule hides unprefixed 'trusted.' attributes on the host. The third rule stops a guest from explicitly setting the 'user.virtiofs.' path directly to prevent access control bypass on the target of the earlier prefix remapping. Finally, the fourth rule lets all remaining attributes through.

This is equivalent to the 'map' rule:

```
-o xattrmap="/map/trusted./user.virtiofs./"
```

3. Hide 'security.' attributes, and allow everything else

```
"/bad/all/security./security./
 /ok/all///'
```

The first rule combines what could be separate client and server rules into a single 'all' rule, matching 'security.' in either client arguments or lists returned from the host. This stops the client seeing any 'security.' attributes on the server and stops it setting any.

## EXAMPLES

Export **/var/lib/fs/vm001/** on vhost−user UNIX domain socket **/var/run/vm001−vhost−fs.sock**:

```
host# virtiofsd −−socket−path=/var/run/vm001−vhost−fs.sock −o source=/var/lib/f
host# qemu−system−x86_64 \
      −chardev socket,id=char0,path=/var/run/vm001−vhost−fs.sock \
      −device vhost−user−fs−pci,chardev=char0,tag=myfs \
      −object memory−backend−memfd,id=mem,size=4G,share=on \
      −numa node,memdev=mem \
      ...
guest# mount −t virtiofs myfs /mnt
```

## AUTHOR

Stefan Hajnoczi <stefanha@redhat.com>, Masayoshi Mizuma <m.mizuma@jp.fujitsu.com>

## COPYRIGHT