# FUSE Design Document

**William Krier**

**Erick Liska**

Revision 0.6 (DRAFT)

May 16, 2009

## Table of Contents

## Illustration Index

# 1 Introduction

This document describes the scope of this project: to deliver a standard virtual file system into the Solaris operating system by porting the FUSE infrastructure from FreeBSD, which will allow existing FUSE file systems to run on Solaris. Specifically, the document describes the user experience, the architecture, the design, the technical problems and resolutions, and the deliverables for the project.

## 1.1 Purpose

This document describes the design of the FUSE (File System in User Space) infrastructure on Solaris. It is intended to contain all the information required to design and develop FUSE for Solaris, including a high level description of the technical issues involved and their proposed solutions.

## 1.2 Intended Audience

The intended audience includes the PSARC review team, the OpenSolaris user community, our Quality Engineering staff and our Technical Writing staff.

## 1.3 References

*FUSE API Specification*

*FUSE Protocol Specification*

*mount(1M)*

*mount_fuse(1M)*

## 1.4 Definitions, Acronyms, and Abbreviations

| Term | Definition |
|---|---|
| FUSE | File system in User Space |
| SSHFS | FUSE file system based on Secure SHell protocol |
| NTFS-3g | FUSE file system based on NTFS |
| DAVfs | FUSE file system based on WebDAV |
| VFS | Virtual File System |
| vnode | Virtual (file system) node |
| RBAC | Role Based Access Control |

.

# 2 Project Overview

## 2.1 Description

FUSE is a framework for implementing file systems in user space, eliminating the need to write kernel modules in order to implement a file system. FUSE originated from the Linux community, and there is a FreeBSD port as well.

FUSE itself has two parts: a kernel module and a user land resident library which provides a framework and application programming interface (API) through which a file system can be implemented entirely within user space.

There are several advantages to FUSE: any programming language can be used for implementing a file system, and the file system author can use all the POSIX and user space libraries, bypassing possible constraints of having a limited API set within the kernel. Also, the file system implementation is isolated from kernel programming and operating system intricacies.

## 2.2 FUSE File Systems

A number of file systems have been developed using the FUSE API, including traditional disk-based file systems such as ntfs, fat, zfs, and ext2. There are also FUSE file systems designed to access files inside archives (rarfs, fuse.gunzip, fuse-zip); file systems to access data in compressed images (LZOlayer_fs, FuseCompress); file systems to access data on remote systems (sshfs, DAVfs); and many more. A more comprehensive list of FUSE file systems can be found here: http://apps.sourceforge.net/mediawiki/fuse/index.php?title=FileSystems.

Since the FUSE infrastructure has no value if there are no FUSE file systems available, this project will include a port of the sshfs file system to Solaris. The sshfs file system is based on the SSH File Transfer Protocol. Since most SSH servers already support this protocol it is very easy to set up; on the server side there is nothing that must be done. On the client side, mounting the file system is as easy as logging into the server with ssh.

sshfs was authored by Miklos Szeredi (miklos at szeredit dot hu), and can be found at http://fuse.sourceforge.net/sshfs.html.

Ports of a read-write NTFS file system, ntfs-3g, and a WebDAV file system, DAVfs, are also planned for the immediate future.

## 2.3 Functional Overview

The FUSE infrastructure consists of two parts: a FUSE kernel module, and a user space FUSE library (*libfuse*). The FUSE kernel module implements a virtual file system (vfs/vnode) interface and a character device interface. The FUSE library provides a framework through an exported API that is used to implement FUSE file systems.

The kernel module is composed of a virtual file system, *fusefs*, and a character device interface exported through the pseudo device driver */dev/fuse*. The kernel module and user space library communicate through the character device; file system requests are received by *fusefs* and transmitted through */dev/fuse* to the library, where they are processed and the results returned back through the device to *fusefs*.

The FUSE file system starts by registering a set of callback functions for FUSE-defined file system operations. These operations are a user space file system abstraction; they are a only a rough match to file system operations defined by the vfs/vnode layers of UNIX-type operating systems (primarily Linux). After the operation callbacks are registered, the file system is mounted.

First, however, */dev/fuse* is opened. The resulting file descriptor is passed in the mount system call to the kernel via a *fusefs* specific mount option. This file descriptor is used by the *fusefs* mount operation to associate the minor device number of the fuse device with the mount point. This association is subsequently used to pass all vfs and vnode operations from *fusefs* to the fuse device; then from there up to *libfuse*.

Once the mount has completed, *libfuse* uses the fuse device file descriptor to read from the device. This call blocks inside the fuse device until data is available from *fusefs*. As file system requests are received through the vfs, they are packed into requests with the proper code identifying the FUSE-defined file system operation and placed in a queue. At this point, the blocked read from *libfuse* is awakened, and the read returns. The library dispatches the request to the proper file system callback. Meanwhile the process which originated the file system request blocks in *fusefs*.

Once the request has been processed by the FUSE file system callback, the reply is sent back to with a write to */dev/fuse*. Once the write to the device completes, the process blocked inside *fusefs* is awakened, the file system operation completes, and the reply is passed back to the caller. Meanwhile, *libfuse* makes another read call to the device, waiting for further requests, and the process repeats.

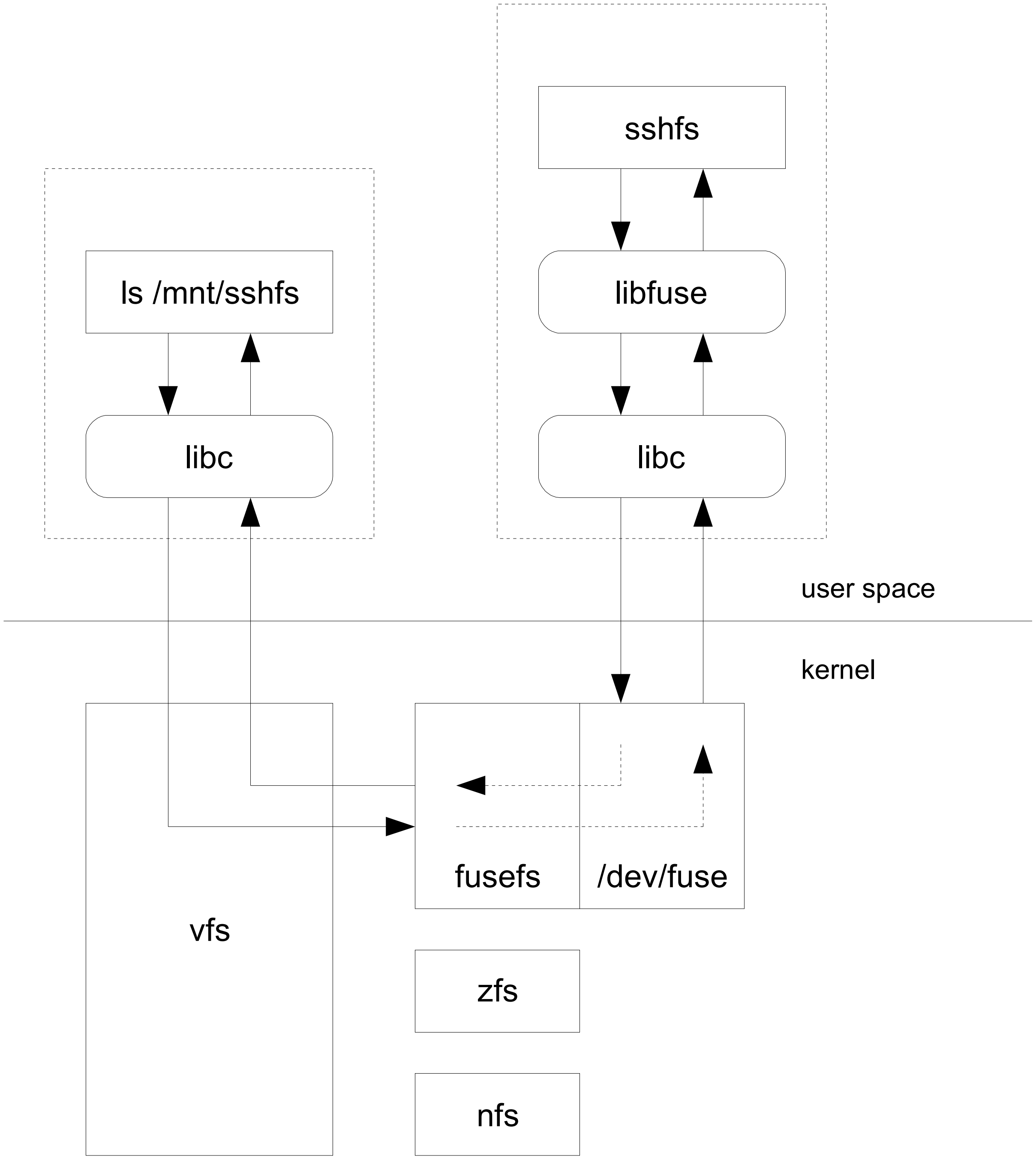

*Illustration 1: Overview of "ls" command on a sshfs file system*

## 2.4 FUSE File System Model

A FUSE file system is simply a user space process that communicates with the FUSE character device. The process performs the file system mount, then forks itself into a daemon and services file system requests from the FUSE character device indefinitely, until the file system is unmounted. Both of these steps are facilitated by the FUSE library; the actual mount, fork, and processing loop are performed by library routines. There is no separation between the mounting of the file system and file system request handling; both are performed by the same program, the resulting processes for which belong to the user who started the program.

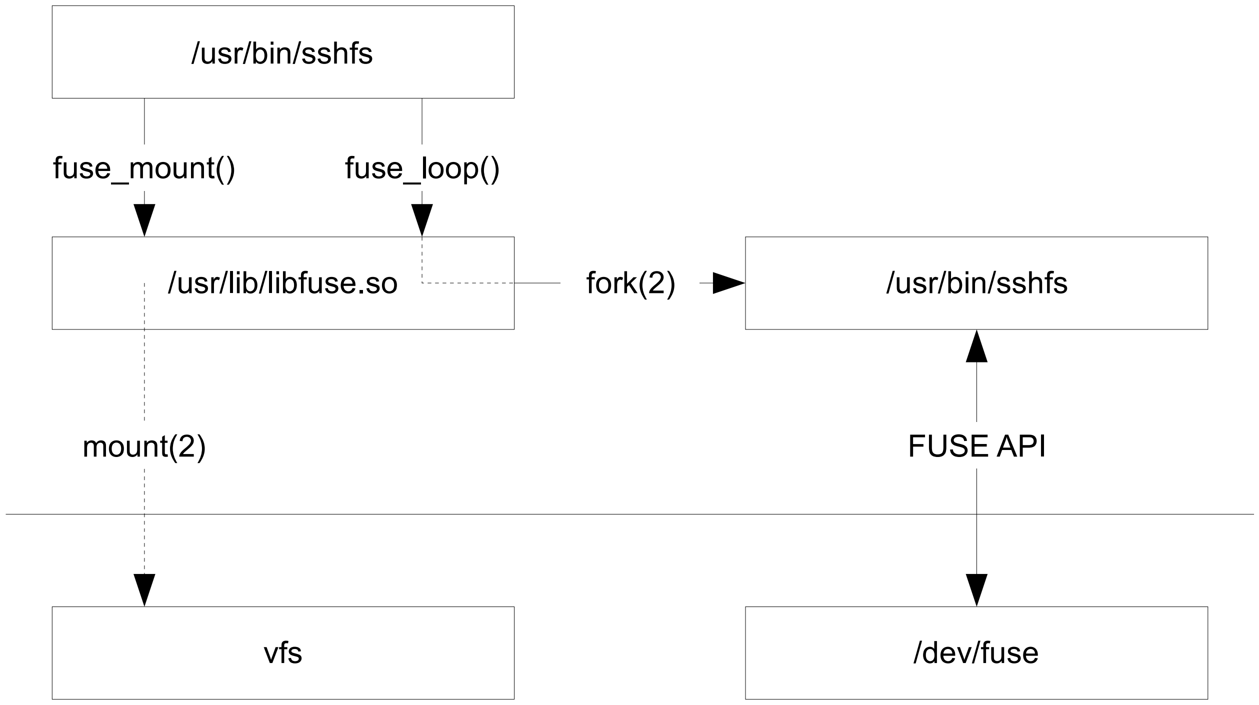

*Illustration 2: Overview of sshfs Process Flow*

The illustration above describes the FUSE file system sshfs; once launched, sshfs calls the *libfuse* function `fuse_mount()` to mount the file system, the calls `fuse_loop()`, which forks sshfs into a daemon process to service file system requests.

By default, FUSE file systems are only accessible to the user who mounted the file system. In this default case, then, the owner of the file system is the only user of the file system. Users other than the owner are not, by default, allowed any access to the files within the file system. Configuration and security implications of this architecture are discussed further in section 4.1.2 File System Access by Other Users.

# 3 Design Considerations

## 3.1 Goals and Guidelines

The primary goals for the FUSE project are functionality, compatibility with existing FUSE file systems, and stability. In addition, development will adhere to the following goals and guidelines:

- Organize design so we can develop with the OpenSolaris community.
- Limit initial functionality to match existing implementations in Linux and FreeBSD so that porting of existing FUSE file systems will be as straight forward as possible.
- Integrate FUSE so that it will work seamlessly with existing Solaris commands.

## 3.2 Development Strategy

Source code from a BSD variant has been chosen as a base for development of the FUSE kernel module for two main reasons: the license is compatible with the CDDL, and the BSD VFS is closer to that in Solaris than the Linux VFS. Specifically, the *fuse4bsd* project, at version *0.3.0-pre1*, was selected as a starting point.

### 3.2.1 Methodology

The FUSE user space library will remain compatible with the original library developed for Linux, which is publicly available. Any modifications and enhancements made to the FUSE library during the port will be offered to the greater FUSE community for inclusion in future releases.

On the other hand, the port of the FUSE kernel module presents some issues. While the BSD port is closer to Solaris than the Linux port, there are still several significant differences between BSD and Solaris both in the VFS as well as other interfaces within the kernel. The intention for FUSE is that it be a maintainable part of Solaris, so that improvements and functional extensions may be added in the future. Along the same lines, the sustaining engineering for FUSE should be as straightforward as possible.

In light of these considerations, no attempt will be made to maintain a code base that is compatible with both BSD and Solaris, say with the use of conditional macros. Instead, we have opted to design a top-level layer that replicates the structure of other Solaris file systems, then port FUSE-specific logic and protocol code into that structure. This will constrain the ability to apply simple code diffs from the BSD code base, but will make it much easier to find home-grown bugs that will be created during the port, and will make the code more approachable for Solaris developers.

### 3.2.2 Re-use and Adaptation of Existing Solaris Source Code

The existing Solaris NFSv2 file system code will be used as a basis for the file system operations portion the kernel module. The NFSv2 implementation is simpler than most other file systems, making it easier to pare down and adapt to FUSE data structures and functionality.

### 3.2.3 Node Operations

The design of the FUSE file system operation callback structure reflects the file-centric nature of the Linux VFS. The FUSE file system operations that correspond to Linux file operations are parameterized by a FUSE-defined file structure, which contains a file handle field. When the `open()` system call is made in Linux, an `OPEN` message is sent to the file system daemon, which performs any necessary processing and sends back a file handle to the kernel. The kernel associates this userland file handle from the daemon directly with its corresponding file structure. In subsequent calls where the kernel file structure is the basis for I/O, the kernel simply retrieves the userland file handle from the kernel file structure, and sends it to the daemon.

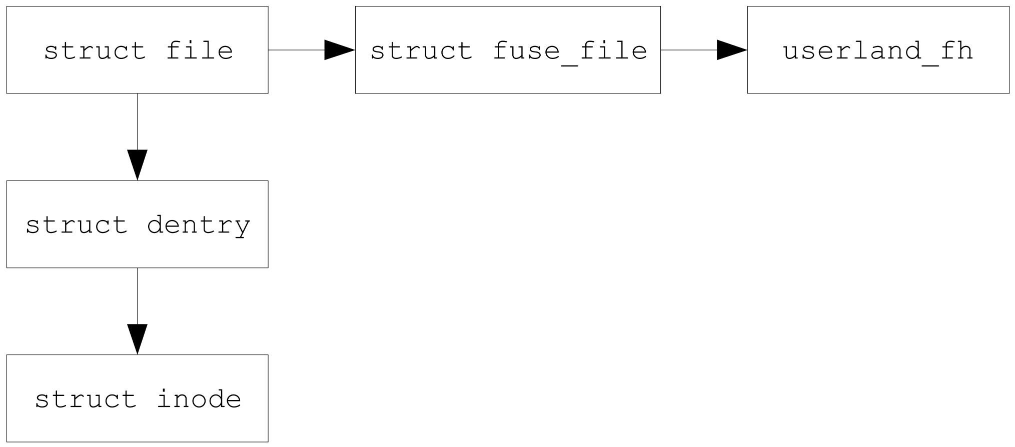

*Illustration 3: Kernel and Userland File Mapping in Linux FUSE*

The userland file handle is treated by the kernel as a cookie; it is associated with a kernel structure representing an open file, but is only actually used by the file system daemon. And, its use is actually not necessary. Indeed, the ntfs-3g file system does not make use of it at all.

FUSE file system operations are fully parameterized; for example, in addition to the userland file handle, a `READ` operation callback accepts a path name, buffer, offset, and size. The daemon file handle is essentially stateless, and only serves to identify the file system node that the daemon is to operate on. So, for the kernel, the only requirement regarding userland file handles is that they remain for the duration of the open state of the file, and that they are stored and retrieved in such a way that they match the file system node with which they were associated when that node was first opened.

In Solaris, as in BSD, the vnode is central to file system operations. Without a kernel-defined file structure as the basis for I/O operations, the relationship between vnodes and userland file handles is not as straightforward as it is in Linux. However, it is not overly complex; file handles are simply "hung off" their associated vnode.

The Solaris FUSE kernel module defines a node context that includes the vnode and FUSE-specific data.

Within this node context is a list of FUSE-defined file handle objects, which contain the userland file handle and data representing the open file, including mode, process credential, and process id.

For each opened file system node, a FUSE file handle object containing information for that node is added to the node context list. On subsequent file system operations, a suitable userland file handle for the specified vnode is retrieved from the context list and sent to the daemon. A "suitable" file handle is defined as one that matches the credential, process id, and mode with which the file was opened. There may be multiple FUSE file handle objects -- and therefore userland file handles -- for a particular vnode, representing multiple `OPEN` operations on that vnode. But any userland file handle with a credential, process id, and mode matching those with the request -- again, for a specific vnode -- will suffice for the daemon because the file handle is stateless, and functions only as a file system node identifier for the daemon.

So, instead of a direct mapping from a kernel file structure to the userland file handle as in Linux, the credential, process id, and mode from the request are compared to those stored with the file handles in the node context list. Any userland file handle matching these conditions is a sufficient identifier for the vnode and can be sent to the daemon with the request.

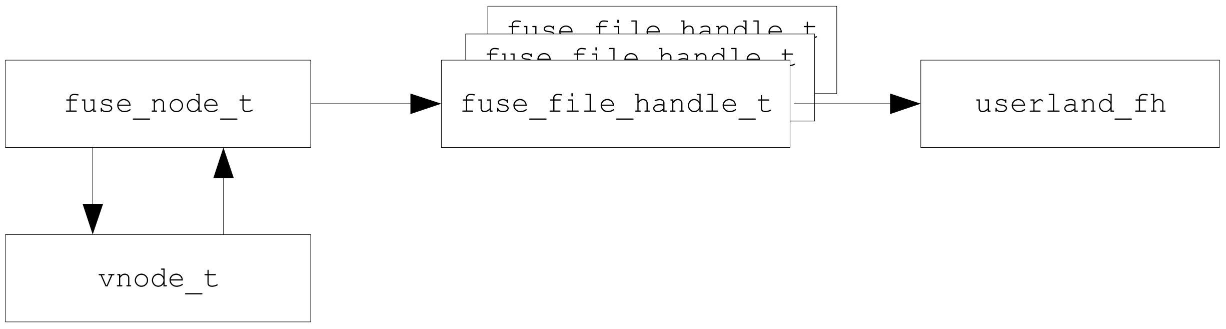

*Illustration 4: Kernel and Userland File Mappings in Solaris FUSE*

### 3.2.4 Integration with Solaris Commands

Most commands relevant to file systems will not require special consideration. However, the mount command will. The FUSE project will include provisions for integration of FUSE file systems with the mount(1M) command, and for support of non-privileged mounts.

#### *3.2.4.1 Integration with mount(1M) Command*

Additions will be made to the file system dependent utility namespace, */usr/lib/fs*, to accommodate mount and unmount helper programs for FUSE file systems:

*/usr/lib/fs/fuse/mount*
*/usr/lib/fs/fuse/umount*

This will allow the mount(1M) command to recognize a file system type of "fuse." The specific FUSE file system sub-type must also be specified in order for the mount program to execute the appropriate file system program. Syntax for mounting FUSE file systems with the mount(1M) command is detailed in mount_fuse(1M).

#### 3.2.4.2 Support for Non-privileged Mounts

Internally, a FUSE-based file system process calls into the FUSE library to mount a file system. The library attempts to mount the file system by making the mount(2) system call. For a root user, this call is sufficient for the mount to succeed (assuming no device or path errors). However, for a typical non-privileged mount, this call will fail due to insufficient privileges. In this case, the task is delegated to a separate FUSE mount program, with the intention that it is configured in such a way that the user's privileges are escalated enough to make the mount system call successfully.

In Linux, the FUSE mount program is called *fusermount*, and is intended to be setuid root (assuming the system is intended to support non-privileged mounts).

In FreeBSD, the program is called *mount_fusefs*; in this case, the model for non-privileged mount support is that the system control variable *vfs.usermount* is set to "1", and the user(s) who can perform non-privileged mounts are given read-write access to the fuse device (typically this is done by adding those users to the operator group, but can also be done with finer-grained control using ACLs).

Solaris affords a tighter integration with the system mount and umount commands than what is possible in Linux and BSD. RBAC facilities obviate the need to make the *fusermount* program setuid root. The FUSE installation will include creation of a unique profile that an administrator can apply to users who are allowed to mount FUSE file systems. The installation will only create this profile, it will not apply this profile to any users.

Added to */etc/security/prof_attr*:

**FUSE File System Management:::Mount and unmount FUSE filesystems:**

Added to */etc/security/exec_attr*:

**FUSE File System Management:solaris:cmd:::/usr/lib/fs/fuse/fusermount.bin:privs=sys_mount**

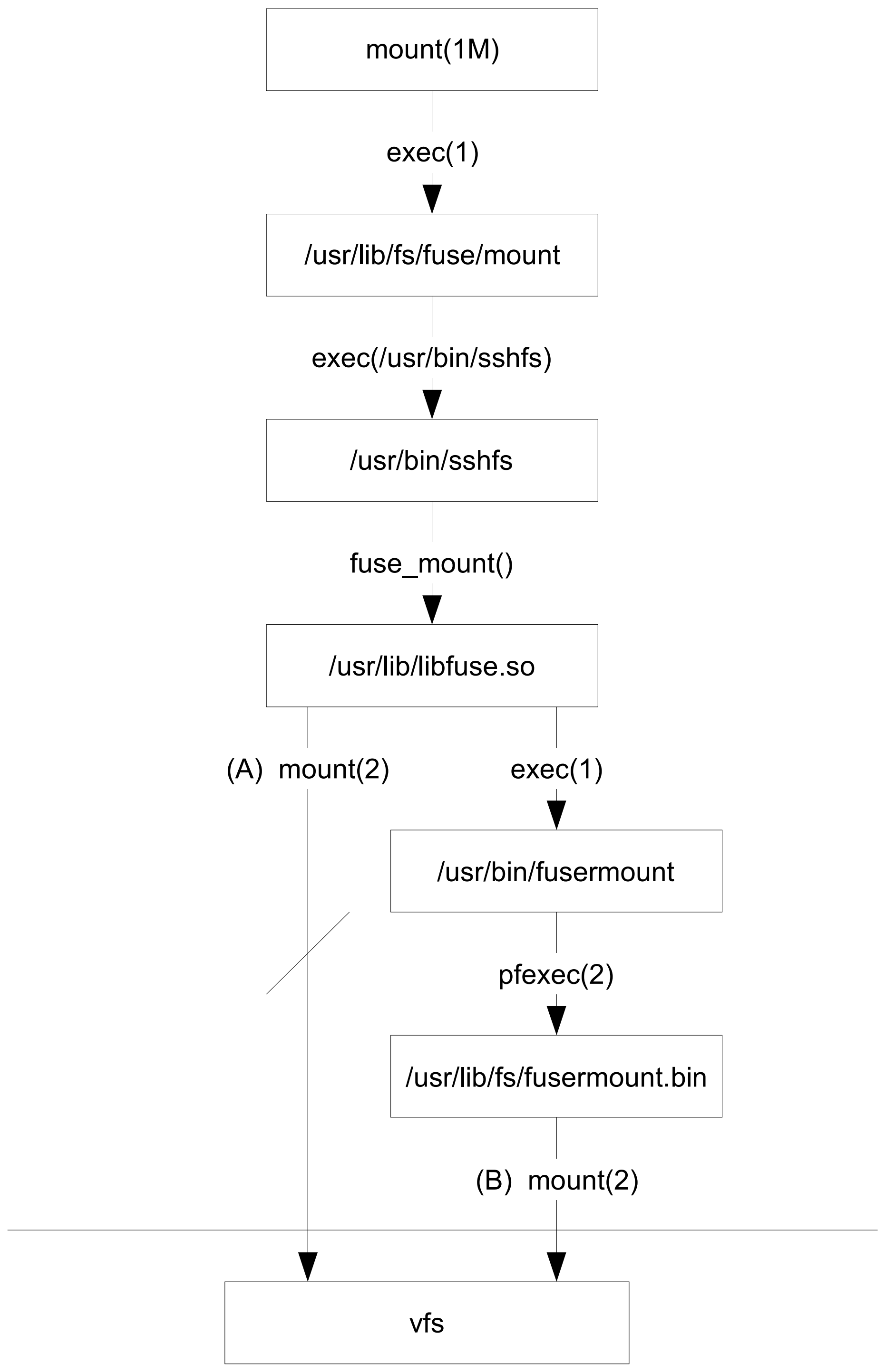

*Illustration 5: Mount of a sshfs File System by Non-Privileged User*

In the illustration above, the first call (A) to mount(2) fails with EPERM. The second call (B) is performed from the external program *fusermount*, which in Linux is setuid root, while in Solaris, *fusermount* has the *sys_mount* privilege attached to it for the FUSE profile. Assuming the user has the FUSE profile, the second call (B) will succeed.

The *fusermount* program on Solaris is implemented as a shell script which executes the binary *fusermount.bin* with the privileges available to the the user (using *pfexec*). The separation between *fusermount* and *fusermount.bin* is primarily to accommodate Linux familiarity; many users use *fusermount* directly to mount and unmount FUSE file systems.

### 3.2.5 Operating System Specifics

#### *3.2.5.1 Memory management policies*

The standard kernel memory allocator calls (`kmem_alloc()` and friends) will be used; the conversion from fuse4bsd's `malloc()` family of calls is mostly obvious.

#### *3.2.5.2 Zones*

Zones is a virtualization solution for Solaris which supports multiple virtual hosts on a single Solaris kernel instance. Kernel modules must be modified to use allocated memory and to ensure that threads are not able to access resources they should not or escalate privileges.

The expectation for FUSE is that it will work the same way in all zones without behavioral differences.

#### *3.2.5.3 Codeset Conversions*

There are a variety of codesets in use by the various file systems including ISO8859, EUC, UTF-8, Unicode and windows code pages to name a few. The FUSE module makes no assumptions as to the codeset in use other than that all filenames passed between the kernel module and the fuse library are NULL terminated strings. It is the responsibility of the file system to do any codeset conversion that may be required.

# 4 Security Considerations

There are two main areas of security that are impacted by FUSE: non-privileged mounts, and FUSE file system access control.

## *4.1 Non-Privileged Mounts*

FUSE is designed to allow non-privileged mounts, if supported by the operating system and administrative policy. On Solaris, FUSE file systems can be mounted by non-privileged users via the RBAC provisions detailed in section 3.2.4.2 (Support for Non-privileged Mounts).

### 4.1.1 Mount Restrictions for Non-Privileged Users

FUSE file systems are mounted with the *nodevices* and *nosetuid* options by default. This can only be overridden by root; non-root users cannot mount a file system with either the *devices* or *setuid* options. This is actually enforced both in the FUSE library and in Solaris by existing operating system security policy processing.

Additionally, non-privileged users must have ownership of the mount point.

### 4.1.2 File System Access by Other Users

By default, FUSE file systems are only accessible to the user who mounted the file system, so that the owner is the only user of the file system.

To review, a FUSE file system is simply a user space daemon; the file system request handling and the file system mount are performed by the same process. Users other than the owner, then, are not by default allowed any access to the files within the file system. If this restriction is overridden, the requests for the FUSE file system from other users' processes are ultimately handled by the daemon owned by the user who mounted the file system.

This prohibition of access by users other than the file system owner prevents a FUSE file system daemon from having the ptrace-like capability of recording operations from another user's process, or allowing a denial of service for another user's process by, say, stalling system calls.

The constraint can be overridden by the mutually exclusive *allow_other* and *allow_root* mount options, which allow processes of other users (including root) and just the root user, respectively, to access a FUSE file system. Note that other users, in this case, need not have the same privileges afforded to the user who originally launched the file system process and thereby mounted the file system.

By default, *allow_other* can only be specified by root (*allow_root* is not applicable to root). For non-root users, the *allow_other* and *allow_root* options cannot be used unless specified in the FUSE configuration file */etc/fuse.conf*.

### 4.1.3 Use of RBAC

FUSE introduces a profile which attaches the *sys_mount* privilege to the *fusermount.bin* program. This program is executed by the FUSE library's mount function as recourse in the case where the mount system call fails because of insufficient privilege, allowing a non-root user with the FUSE profile to make the mount system call successfully. Note that FUSE only creates the profile, it does not apply it to any users.

While a non-root user may have enough privileges to successfully execute the mount system call, the mount may still be disallowed by the file system for other reasons. One such reason may be insufficient access to a block device on which the file system resides.

#### 4.1.4 Block Device Access

For FUSE file systems which can be backed by a block device, such as ntfs-3g, the file system must have read-write access to the device. This amounts to the file system daemon needing read-write access to the block device, which for non-root users is not the case by default. Note that this is a separate authorization requirement from that needed to actually mount the file system, as the file system daemon will need to open the device itself, typically before the mount is attempted.

On Linux, block device-backed file systems, such as ntfs-3g, are simply denied access to the block device, unless the permissions of the device have been changed so that everyone has write permission. BSD offers finer grained control with ACLs.

Another way of dealing with block device access for non-privileged users is to make the file system program setuid root. However, this is risky as it results in privilege escalation that may have unanticipated results, and is generally discouraged. If ntfs-3g is setuid root, for example, it will not run unless it was built using its own internal version of the FUSE library, rather than the typical separate FUSE shared object implementation.

On Solaris, block device access is simply refused to non-privileged users; this is the default configuration. It is entirely up to the system administrator to change this if desired; FUSE makes no assumptions or changes to the system in regards to block device access.

## *4.2 FUSE File System Access Control*

By default, FUSE leaves all access control to the file system or the underlying access method, for example the ssh protocol in the case of sshfs. The mount option *default_permissions* enables FUSE to perform access control based on the UNIX permissions that are presented by the file system. The file system itself is responsible for extrapolating access control into a traditional UNIX permission set, against which FUSE checks the user's credentials.

There are two main limitations to this: it only allows UNIX-style permission checking, bypassing more sophisticated access controls which may be present in the file system; and it does not have direct access to file permissions, as it can only see what the file system itself presents.

The disposition of file ownership varies by file system. For example, ntfs-3g by default presents the owner and group for all files as those of the user who mounted the file system. This can be overridden with mount options (*uid=n, gid=n*) but the user and group specified will apply to all objects on the file system.

On the other hand, sshfs presents file ownership of existing files as the remote system reports them, and can enforce on its own any permission checking performed on the remote host. New files are owned by the user specified in the mount (either in the "device" string -- format *user@host*: -- or with the *uid=n* option).

## *4.3 FUSE Security Configuration*

### 4.3.1 Security-Related Mount Options

The following table contains a summary of security-related mount options for FUSE. None of the options are specified by FUSE by default. However, some FUSE file systems may specify one or more options by default. An example is ntfs-3g; by default it specifies the *allow_other* and *default_permissions* options.

| **Option** | **Function** | **Notes** |
|---|---|---|
| *allow_other* | Permit access to file system from other users' processes. | Cannot be specified by non-root user unless explicitly allowed in /etc/fuse.conf. |
| *allow_root* | Permit access to file system from root processes. | Not applicable to root. Cannot be specified by non-root user unless explicitly allowed in /etc/fuse.conf. |
| *suid/devices/setuid* | Allow access to special files/allow setuid or setgid execution. | Cannot be specified by non-root user; enforced in both libfuse and OS security policy. |
| *default_permissions* | Enables permission checking by FUSE, restricting file access based on file mode. | If not specified, access control is delegated completely to the file system or underlying access mechanism. |
| *uid/gid* | Overrides the st_uid/st_gid stat fields set by file system, assigning specified user or group as owner of files. | |
| *umask/fmask/dmask* | Overrides permission bits in the st_mode stat field set by file system, assigning permission bits as the ones missing from the given mask value for all files/only files/only directories. | |

*Table 1: FUSE Security-Related Mount Options*

### 4.3.2 The FUSE Configuration File

The FUSE configuration file, */etc/fuse.conf,* does not exist by default. It currently supports two options:

*mount_max=N*

Sets the maximum number of FUSE mounts allowed for non-root users. Defaults to 1000.

*user_allow_other*

Allows non-root users to specify the *allow_other* or *allow_root* mount options.

# 5 Deliverables

## *5.1 Package Names*

The FUSE service will be delivered in the following packages.

| | |
|---|---|
| ● SUNWfusefs: | FUSE kernel module package |
| ● SUNWlibfuse: | FUSE library package |

## *5.2 Package Dependencies*

- SUNWlibfuse has a dependency on the SUNWfusefs package being installed.

## *5.3 Package Contents*

### 5.3.1 SUNWfusefs

Components installed on all platforms

1. **usr/kernel/drv/fuse.conf**
   File Permissions: 644
   Owner: root
   Group: sys
   Desc: Kernel fusefs driver configuration file

Components installed specific to x86 platforms

1. **usr/kernel/drv/fuse**
   File Permissions: 755
   Owner: root
   Group: sys
   Desc: Kernel fusefs driver binary for 32-bit x86 platform

1. **usr/kernel/drv/amd64/fuse**
   File Permissions: 755
   Owner: root
   Group: sys

Desc: Kernel fusefs driver binary for 64-bit x86 platform

Components installed specific to sparc platforms

1. **usr/kernel/drv/sparcv9/fuse**
   File Permissions: 755
   Owner: root
   Group: sys
   Desc: Kernel fusefs driver binary for 64-bit sparc platform

### 5.3.2 SUNWlibfuse

Components installed on all platforms

1. **usr/lib/pkgconfig/fuse.pc**
   File Permissions: 444
   Owner: root
   Group: bin
   Desc: Package configuration file for FUSE lib.
2. **usr/lib/fs/fuse**
   File Permissions: 755
   Owner: root
   Group: sys
   Desc: FUSE file system-dependent subdirectory
3. **usr/lib/fs/fuse/mount**
   Permissions: 755
   Owner: root
   Group: bin
   Desc: FUSE file system-dependent mount program
4. **usr/lib/fs/fuse/umount**
   Permissions: 755
   Owner: root
   Group: bin
   Desc: FUSE file system-dependent unmount program
5. **usr/include/fuse/fuse.h**
   **usr/include/fuse/fuse_common.h**
   **usr/include/fuse/fuse_common_compat.h**
   **usr/include/fuse/fuse_compat.h**
   File Permissions: 644
   Owner: root
   Group: bin
   Desc: FUSE header files defining statndard FUSE API
6. **usr/include/fuse/fuse_lowlevel.h**
   **usr/include/fuse/fuse_lowlevel_compat.h**

File Permissions: 644
Owner: root
Group: bin
Desc: FUSE header files defining lowlevel FUSE API

7. **usr/include/fuse/fuse_opt.h**
   File Permissions: 644
   Owner: root
   Group: bin
   Desc: FUSE header file defining the options parsing for FUSE.

Components installed specific to x86 platforms

1. **usr/lib/libfuse.so.2.7.1**
   File Permissions: 755
   Owner: root
   Group: bin
   Desc: FUSE library for 32-bit x86 platforms
2. **usr/lib/libfuse.so**
   File Permissions: 777
   Owner: root
   Group: root
   Desc: symbolic link to FUSE library for 32-bit x86 platforms
3. **usr/lib/amd64/libfuse.so.2.7.1**
   File Permissions: 755
   Owner: root
   Group: bin
   Desc: FUSE library for 64-bit x86 platforms
4. **usr/lib/amd64/libfuse.so**
   File Permissions: 777
   Owner: root
   Group: root
   Desc: symbolic link to FUSE library for 64-bit x86 platforms
5. **usr/bin/fusermount**
   File Permissions: 755
   Owner: root
   Group: bin
   Desc: executable shell script to launch fusermount.bin
6. **usr/lib/fs/fuse/fusermount.bin**
   File Permissions: 755
   Owner: root
   Group: bin
   Desc:  helper application to mount FUSE file systems

7. **usr/bin/amd64/fusermount**
   File Permissions: 755
   Owner: root
   Group: bin
   Desc:  user executable shell script to launch fusermount.bin
8. **usr/lib/fs/fuse/amd64/fusermount.bin**
   File Permissions: 755
   Owner: root
   Group: bin
   Desc: helper application to mount FUSE file systems for 64-bit x86 platforms
9. **usr/lib/fs/fuse/mount**
   File Permissions: 755
   Owner: root
   Group: bin
   Desc: fuse specific mount program
10. **usr/lib/fs/fuse/amd64/mount**
    File Permissions: 755
    Owner: root
    Group: bin
    Desc: fuse specific mount program for 64-bit x86 platforms
11. **usr/lib/fs/fuse/umount**
    File Permissions: 755
    Owner: root
    Group: bin
    Desc: fuse specific unmount program
12. **usr/lib/fs/fuse/amd64/mount**
    File Permissions: 755
    Owner: root
    Group: bin
    Desc: fuse specific mount program for 64-bit x86 platforms

Components installed specific to sparc platforms

1. **usr/lib/sparcv9/libfuse.so.2.7.1**
   File Permissions: 755
   Owner: root
   Group: bin
   Desc: FUSE library for 64-bit sparc platforms
2. **usr/lib/sparcv9/libfuse.so**
   File Permissions: 755
   Owner: root
   Group: root

Desc: symbolic link to FUSE library for sparc platforms

3. **usr/bin/sparcv9/fusermount**
   File Permissions: 755
   Owner: root
   Group: bin
   Desc: executable shell script to launch fusermount.bin
4. **usr/lib/fs/fuse/sparcv9/fusermount.bin**
   File Permissions: 755
   Owner: root
   Group: bin
   Desc: helper application to mount FUSE file systems for sparc platforms

# 6 Interfaces

## 6.1 Exported Interfaces

| Interface | Classification | Comments |
|---|---|---|
| FUSE API | Committed | Version 2.7 of specification, detailed below (section 8.2) |
| FUSE protocol | Project Private | Version 7.8 of specification, detailed below (section 8.3) |
| fusefs module | Consolidation Private | VFS file system operations |

The FUSE API defines the interface used by developers to implement FUSE file systems in user space. The FUSE API interface is classified as Committed. Although there are past indications that the controlling body has made efforts to maintain backward compatibility, there is no guarantee that this would be the case going forward. Therefore every effort will be made to maintain compatibility with version 2.7 of the API. Version 2.7 of the FUSE API is summarized in section 8 and a more detailed description of the the FUSE API can be found in the *FUSE API Specification*.

The FUSE protocol defines the communication between the FUSE kernel module and the FUSE user space library. This protocol is classified as Project Private. A detailed description of the FUSE protocol can be found in the *FUSE Protocol Specification*.

## 6.2 Imported Interfaces

| Interface | Classification | Comments |
|---|---|---|
| fop_getattr | Consolidation Private | VOP_GETATTR |
| fop_inactive | Consolidation Private | VOP_INACTIVE |
| fop_lookup | Consolidation Private | VOP_LOOKUP |
| fop_mkdir | Consolidation Private | VOP_MKDIR |
| fop_putpage | Consolidation Private | VOP_PUTPAGE |
| fop_setattr | Consolidation Private | VOP_SETATTR |
| vn_alloc | Consolidation Private | |
| vn_free | Consolidation Private | |
| vn_freevnodeops | Consolidation Private | |
| vn_has_cached_data | Consolidation Private | |
| vn_make_ops | Consolidation Private | |
| vn_matchops | Consolidation Private | |
| vn_mountedvfs | Consolidation Private | |

| Interface | Classification | Comments |
|---|---|---|
| vn_rele | Consolidation Private | VN_RELE |
| vn_setops | Consolidation Private | |
| vn_vfsunlock | Consolidation Private | |
| vn_vfswlock | Consolidation Private | |
| dounmount | Consolidation Private | sys/vfs.h |
| fsop_root | Consolidation Private | VFS_ROOT |
| vfs_freevfsops_by_type | Consolidation Private | |
| vfs_hold | Consolidation Private | |
| vfs_make_fsid | Consolidation Private | |
| vfs_optionisset | Consolidation Private | |
| vfs_rele | Consolidation Private | |
| vfs_setfsops | Consolidation Private | |
| bp_mapin | Consolidation Private | sys/buf.h |
| bp_mapout | Consolidation Private | |
| pageio_done | Consolidation Private | |
| pageio_setup | Consolidation Private | |
| crfree | Consolidation Private | sys/cred.h |
| crhold | Consolidation Private | |
| groupmember | Consolidation Private | |
| getf | Consolidation Private | sys/file.h |
| releasef | Consolidation Private | |
| as_map | | vm/as.h |
| as_rangelock | | |
| as_rangeunlock | Consolidation Private | |
| pvn_getdirty | Consolidation Private | vm/pvn.h |
| pvn_getpages | Consolidation Private | |
| pvn_plist_init | Consolidation Private | |
| pvn_read_done | Consolidation Private | |
| pvn_vplist_dirty | Consolidation Private | |
| pvn_write_done | Consolidation Private | |
| segmap_getmapflt | Consolidation Private | |

| Interface | Classification | Comments |
|---|---|---|
| segmap_pagecreate | Consolidation Private | |
| segmap_pageunlock | Consolidation Private | |
| segmap_release | Consolidation Private | |
| segvn_create | Consolidation Private | vm/seg_vn.h |
| vpm_data_copy | Consolidation Private | vm/vpm.h |
| vpm_sync_pages | Consolidation Private | |

# 7 Detailed System Design

## 7.1 *fusefs*

### 7.1.1 Purpose

*fusefs* is a Solaris virtual file system module with the vfsops and vnodeops. It will be patterned on the NFS Version 2 code in structure, and the FUSE-specific behavior will be ported in from fuse4bsd. It interfaces with the user space fuse library via a character device (*/dev/fuse*). This character device is also part of the fusefs kernel module with the implementation of the cb_ops.

### 7.1.2 Interfaces

#### *7.1.2.1 VFS Operations*

| Solaris VFS | fusefs function | Comment |
|---|---|---|
| vfs_mount | fuse_mount | |
| vfs_unmount | fuse_unmount | |
| vfs_root | fuse_root | |
| vfs_statvfs | fuse_statvfs | |
| vfs_sync | fs_sync | |
| vfs_vget | | |
| vfs_mountroot | | |
| vfs_freevfs | | |
| vfs_vnstate | | |

#### 7.1.2.2 Vnodeops

| Solaris VNODE op | fusefs function | Comment |
|---|---|---|
| vop_open | fuse_open | |
| vop_close | fuse_close | |
| vop_read | fuse_read | |
| vop_write | fuse_write | |
| vop_ioctl | | |
| vop_setfl | | |
| vop_getattr | fuse_getattr | |
| vop_setattr | fuse_setattr | |
| vop_access | fuse_access | |
| vop_lookup | fuse_lookup | |
| vop_create | fuse_create | |
| vop_remove | fuse_remove | |
| vop_link | fuse_link | |
| vop_rename | fuse_rename | |
| vop_mkdir | fuse_mkdir | |
| vop_rmdir | fuse_rmdir | |
| vop_readdir | fuse_readdir | |
| vop_symlink | fuse_symlink | |
| vop_readlink | fuse_readlink | |
| vop_fsync | fuse_fsync | |
| vop_inactive | fuse_inactive | |
| vop_fid | | |
| vop_rwlock | fuse_rwlock | |
| vop_rwunlock | fuse_rwunlock | |
| vop_seek | fuse_seek | |
| vop_cmp | | |
| vop_frlock | | |
| vop_space | fuse_space | |
| vop_realvp | | |

| Solaris VNODE op | fusefs function | Comment |
|---|---|---|
| vop_getpage | fuse_getpage | |
| vop_putpage | fuse_putpage | |
| vop_map | fuse_map | |
| vop_addmap | fuse_addmap | |
| vop_delmap | fuse_delmap | |
| vop_poll | | |
| vop_dump | | |
| vop_pathconf | | |
| vop_pageio | | |
| vop_dumpctl | | |
| vop_dispose | fuse_dispose | |
| vop_setsecattr | | |
| vop_getsecattr | | |
| vop_shrlock | | |
| vop_vnevent | | |

#### 7.1.2.3 cb_ops

| Solaris CB op | fusefs function | Comment |
|---|---|---|
| cb_open | fuse_dev_open | |
| cb_close | fuse_dev_close | |
| cb_strategy | nodev | |
| cb_print | nodev | |
| cb_dump | nodev | |
| cb_read | fuse_dev_read | |
| cb_write | fuse_dev_write | |
| cb_ioctl | nodev | |
| cb_devmap | nodev | |
| cb_mmap | nodev | |
| cb_segmap | nodev | |

| Solaris CB op | fusefs function | Comment |
|---|---|---|
| cb_chpoll | fuse_dev_poll | |
| cb_prop_op | fuse_dev_prop_op | |
| cb_str | NULL | |
| cb_flag | D_NEW \| D_MP | |
| cb_rev | CB_REV | |
| cb_aread | nodev | |
| cb_awrite | nodev | |

## 7.2 *libfuse*

### 7.2.1 Purpose

The FUSE library (libfuse.so) provides a framework and exports the API used to implement FUSE file systems in userspace.

This section summarizes the exported interfaces of version 2.7 of the FUSE API. A detailed description of the FUSE API can be found in the *FUSE API Specification*.

### 7.2.2 Interfaces

| Interface | Classification | Type | Comments |
|---|---|---|---|
| struct fuse_operations | Committed | data structure | fuse.h |
| struct fuse_context | Committed | data structure | fuse.h |
| struct fuse_module | Committed | data structure | fuse.h |
| struct fuse_file_info | Committed | data structure | fuse_common.h |
| struct fuse_conn_info | Committed | data structure | fuse_common.h |
| struct fuse_entry_param | Committed | data structure | fuse_lowlevel.h |
| struct fuse_ctx | Committed | data structure | fuse_lowlevel.h |
| struct fuse_lowlevel_ops | Committed | data structure | fuse_lowlevel.h |
| struct fuse_session_ops | Committed | data structure | fuse_lowlevel.h |
| struct fuse_chan_ops | Committed | data structure | fuse_lowlevel.h |
| struct fuse_opt | Committed | data structure | fuse_opt.h |
| struct fuse_args | Committed | data structure | fuse_opt.h |
| fuse_main | Committed | define | fuse.h |
| FUSE_REGISTER_MODULE | Committed | define | fuse.h |

| Interface | Classification | Type | Comments |
|---|---|---|---|
| FUSE_MAJOR_VERSION | Committed | define | fuse_common.h |
| FUSE_MINOR_VERSION | Committed | define | fuse_common.h |
| FUSE_ROOT_ID | Committed | define | fuse_lowlevel.h |
| FUSE_OPT_KEY | Committed | define | fuse_opt.h |
| FUSE_OPT_END | Committed | define | fuse_opt.h |
| FUSE_ARGS_INIT | Committed | define | fuse_opt.h |
| FUSE_OPT_KEY_OPT | Committed | define | fuse_opt.h |
| FUSE_OPT_KEY_NOOPT | Committed | define | fuse_opt.h |
| FUSE_OPT_KEY_KEEP | Committed | define | fuse_opt.h |
| FUSE_OPT_KEY_DISCARD | Committed | define | fuse_opt.h |
| fuse_fill_dir_t | Committed | typedef | fuse.h |
| fuse_processor_t | Committed | typedef | fuse.h |
| fuse_ino_t | Committed | typedef | fuse_lowlevel.h |
| fuse_req_t | Committed | typedef | fuse_lowlevel.h |
| fuse_interrupt_func_t | Committed | typedef | fuse_lowlevel.h |
| fuse_opt_proc_t | Committed | typedef | fuse_lowlevel.h |
| fuse_new | Committed | function | fuse.h |
| fuse_destroy | Committed | function | fuse.h |
| fuse_loop | Committed | function | fuse.h |
| fuse_exit | Committed | function | fuse.h |
| fuse_loop_mt | Committed | function | fuse.h |
| fuse_get_context | Committed | function | fuse.h |
| fuse_interrupted | Committed | function | fuse.h |
| fuse_invalidate | Committed | function | fuse.h |
| fuse_main_real | Committed | function | fuse.h |
| fuse_fs_new | Committed | function | fuse.h |
| fuse_register_module | Committed | function | fuse.h |
| fuse_setup | Committed | function | fuse.h |
| fuse_teardown | Committed | function | fuse.h |
| fuse_read_cmd | Committed | function | fuse.h |
| fuse_process_cmd | Committed | function | fuse.h |

| Interface | Classification | Type | Comments |
|---|---|---|---|
| fuse_loop_mt_proc | Committed | function | fuse.h |
| fuse_exited | Committed | function | fuse.h |
| fuse_set_getcontext_func | Committed | function | fuse.h |
| fuse_get_session | Committed | function | fuse.h |
| fuse_mount | Committed | function | fuse_common.h |
| fuse_unmount | Committed | function | fuse_common.h |
| fuse_parse_cmdline | Committed | function | fuse_common.h |
| fuse_daemonize | Committed | function | fuse_common.h |
| fuse_version | Committed | function | fuse_common.h |
| fuse_set_signal_handlers | Committed | function | fuse_common.h |
| fuse_remove_signal_handlers | Committed | function | fuse_common.h |
| fuse_opt_parse | Committed | function | fuse_opt.h |
| fuse_opt_add_opt | Committed | function | fuse_opt.h |
| fuse_opt_add_arg | Committed | function | fuse_opt.h |
| fuse_opt_insert_arg | Committed | function | fuse_opt.h |
| fuse_opt_free_args | Committed | function | fuse_opt.h |
| fuse_opt_match | Committed | function | fuse_opt.h |
| fuse_reply_err | Committed | function | fuse_lowlevel.h |
| fuse_reply_none | Committed | function | fuse_lowlevel.h |
| fuse_reply_entry | Committed | function | fuse_lowlevel.h |
| fuse_reply_create | Committed | function | fuse_lowlevel.h |
| fuse_reply_attr | Committed | function | fuse_lowlevel.h |
| fuse_reply_readlink | Committed | function | fuse_lowlevel.h |
| fuse_reply_open | Committed | function | fuse_lowlevel.h |
| fuse_reply_write | Committed | function | fuse_lowlevel.h |
| fuse_reply_buf | Committed | function | fuse_lowlevel.h |
| fuse_reply_iov | Committed | function | fuse_lowlevel.h |
| fuse_reply_statfs | Committed | function | fuse_lowlevel.h |
| fuse_reply_xattr | Committed | function | fuse_lowlevel.h |
| fuse_reply_lock | Committed | function | fuse_lowlevel.h |
| fuse_reply_bmap | Committed | function | fuse_lowlevel.h |

| Interface | Classification | Type | Comments |
|---|---|---|---|
| fuse_add_direntry | Committed | function | fuse_lowlevel.h |
| fuse_req_userdata | Committed | function | fuse_lowlevel.h |
| fuse_req_ctx | Committed | function | fuse_lowlevel.h |
| fuse_req_interrupt_func | Committed | function | fuse_lowlevel.h |
| fuse_req_interrupted | Committed | function | fuse_lowlevel.h |
| fuse_lowlevel_new | Committed | function | fuse_lowlevel.h |
| fuse_session_new | Committed | function | fuse_lowlevel.h |
| fuse_session_add_chan | Committed | function | fuse_lowlevel.h |
| fuse_session_remove_chan | Committed | function | fuse_lowlevel.h |
| fuse_session_next_chan | Committed | function | fuse_lowlevel.h |
| fuse_session_process | Committed | function | fuse_lowlevel.h |
| fuse_session_destroy | Committed | function | fuse_lowlevel.h |
| fuse_session_exit | Committed | function | fuse_lowlevel.h |
| fuse_session_reset | Committed | function | fuse_lowlevel.h |
| fuse_session_exited | Committed | function | fuse_lowlevel.h |
| fuse_session_loop | Committed | function | fuse_lowlevel.h |
| fuse_session_loop_mt | Committed | function | fuse_lowlevel.h |
| fuse_chan_new | Committed | function | fuse_lowlevel.h |
| fuse_chan_fd | Committed | function | fuse_lowlevel.h |
| fuse_chan_bufsize | Committed | function | fuse_lowlevel.h |
| fuse_chan_data | Committed | function | fuse_lowlevel.h |
| fuse_chan_session | Committed | function | fuse_lowlevel.h |
| fuse_chan_recv | Committed | function | fuse_lowlevel.h |
| fuse_chan_send | Committed | function | fuse_lowlevel.h |
| fuse_chan_destroy | Committed | function | fuse_lowlevel.h |

## 8 Acknowledgements

Henk, Csaba, and Miklos Szeredi. "SourceForge.net: Filesystem in Userspace."
SourceForge.net: Opensource Software. 14 Oct. 2004. 8 Dec. 2008
<http://sourceforge.net/projects/fuse/>

FUSE: Filesystem in Userspace 8 Dec. 2008
<http://fuse.sourceforge.net/>.

"SourceForge.net: Main Page - fuse."
SourceForge.net: Open Source Software. 12 Nov. 2008. 03 Feb. 009
<http://apps.sourceforge.net/mediawiki/fuse>

Mistry, Anish. Fuse for FreeBSD. 19 June 2007. 03 Feb. 2009
<http://fuse4bsd.creo.hu/>.

Rydberg, Johan. "Propp." 9 Nov. 2004. 26 Jan. 2009
<http://www.night.dataphone.se/~jrydberg/propp.html>.

Wolfe, Christopher. "FUSE 7.8 Protocol for Linux."
School of Computing, Queen's University. 17 Sept. 2007. 26 Jan. 2009
<http://research.cs.queensu.ca/~wolfe/misc/rtbfs/fuse-7.8.xml>.