

Fuse Extension

version 0.50

Erick Galesio

Université de Nice - Sophia Antipolis
930 route des Colles, BP 145
F-06903 Sophia Antipolis, Cedex
France

This document was produced using the Scribe Programming Language and its ConTEXt engine.

For further information on Scribe, see <http://www-sop.inria.fr/mimosa/fp/Scribe/>.

Document created on January 15, 2006.

1 Introduction

This extension permits the implementation of virtual file systems in Scheme thanks to the **FUSE library**. Here is an excerpt of the FUSE site:

With FUSE it is possible to implement a fully functional filesystem in a userspace program. Features include:

- *Simple library API*
- *Simple installation (no need to patch or recompile the kernel)*
- *Secure implementation*
- *Userspace - kernel interface is very efficient*
- *Usable by non privileged users*
- *Runs on Linux kernels 2.4.X and 2.6.X*
- *Has proven very stable over time*

2 Compiling Fuse library for Boehm GC

The Fuse package uses Posix threads. As a consequence, you need to compile the Fuse library in a special way (the fuse kernel module doesnt need to be recompiled). Since, the Boehm GC used for STklos GC needs to scan the stacks of each thread, you have to include the “gc.h” in each source file of the “lib” directory. A simple way to achieve this consists to add the line

```
#include <gc.h>
```

in the file “include/config.h”.

Once the “Fuse” library is compiled and installed, the Garbage Collector is aware of all the threads of your program and everything should work. In this is not the case, you should see GC complaints about areas which are not freed in the ththread which allocate them.

3 Using the fuse extension

To use this extension you need to include the following form in your program:

```
(require "fuse")
```

This library provides only one entry point called “fuse-mount”. This function takes a first parameter which is the list of the program arguments and key-list of functions used to implement the file system. The arguments recognized by the version 2.4.1 “fuse-mount” are given below:

```

    FUSE options:
    -d                enable debug output (implies -f)
    -f                foreground operation
    -s                disable multi-threaded operation
    -r                mount read only (equivalent to '-o ro')
    -o opt,[opt...]  mount options
    -h                print help

    Mount options:
    default_permissions  enable permission checking
    allow_other          allow access to other users
    allow_root          allow access to root
    kernel_cache         cache files in kernel
    large_read           issue large read requests (2.4 only)
    direct_io           use direct I/O
    max_read=N          set maximum size of read requests
    hard_remove         immediate removal (don't hide files)
    debug               enable debug output
    fsname=NAME         set filesystem name in mtab
    use_ino             let filesystem set inode numbers
    readdir_ino         try to fill in d_ino in readdir
    nonempty            allow mounts over non-empty file/dir
    umask=M             set file permissions (octal)
    uid=N               set file owner
    gid=N               set file group

```

The list of functions used to implement the file system is given in the next section

4 File system primitives

The following functions are available to implement a Scheme file system:

:getattr	<i>path</i>	returns a vector of 8 elements containing: mode bits, number of links, size, uid, gid, atime, mtime, ctime
:opendir	<i>path</i>	this is a hook for controlling directory access, returns 0 if no error
:readdir	<i>path</i>	returns a list of the files in the firectorry "path"
:releasedir	<i>path</i>	This is a hook called after readdir, returns 0 if no error
:mknod	<i>path mode</i>	creates the file named "path" with given "mode"
:open	<i>path mode fd</i>	opens file named "path" with given mode (O = RDONLY, 1 = WRONLY, 2 = RDWR). The value "fd" is an unique integer associated by the system to this file

:read	<i>fd size offset</i>	returns a string of "size" bytes starting at "offset" on "fd"
:write	<i>fd buffer size offset</i>	writes the first "size" characters of "buffer" at "offset" on "fd". The offset can be after the actual end of file
:release	<i>fd</i>	This function is called when there are no more references to the open file "fd". The return value of this function is ignored
:rename	<i>from to</i>	renames file "from" with name "to"
:unlink	<i>path</i>	removes the file with given "path"
:link	<i>old new</i>	creates a link from file "old" to file "to"
:symlink	<i>old new</i>	creates a symbolic link from file "old" to file "to"
:readlink	<i>path</i>	returns the file that the symbolic link "path" point to
:mkdir	<i>path</i>	creates directory "path"
:rmdir	<i>path</i>	removes directory "path"
:chmod	<i>path mode</i>	change the mode of file "path" to "mode"
:chown	<i>path uid gid</i>	changes the owner of "path" to "uid" and "gid"
:utime	<i>path atime mtime</i>	changes the access and modification time of file "path" to "atime" and "mtime"
:truncate	<i>path size</i>	changes the size of "path" to "size"
:flush	<i>path fd</i>	flushes cached data on file "fd"
:fsync	<i>path datasync fd</i>	if the "datasync" parameter is non-zero, then only the user data should be flushed, not the meta data
:fsyncdir	<i>path datasync</i>	if the "datasync" parameter is non-zero, then only the user data should be flushed, not the meta data
:init		This is a hook called when the file system is mounted. It can return a value which will be used when the file system is unmounted
:destroy	<i>data</i>	This is a hook called when the file system is unmounted. Its parameter is the return value of the "init" call.

5 The hellofs filesystem

The following example is a very simple (even simplistic) file system written in STKLOS. This is a file system which contains only a file named "hello". You cannot do a lot with this file system and most actions produce errors. To mount the file system you can for instance type:

```
$ hellofs -f ~/fuse
```

This will mount the hellofs on the (already existing and empty) "~/fuse" directory. To unmount this file system, you can do:

```
$ fusermount -u ~/fuse
```

A more complete and realistic example is provided in the "examples" directory.

```

(require "posix")
(require "fuse")

(define content "Hello, world!\n")

(define (main args)
  (fuse-mount args
    :getattr (lambda (path)
      (let ((tm (current-time)))
        (cond
          ((equal? path "/")
           (vector (+ posix/IFDIR #o755) ;; mode
                   2 ;; links
                   123 ;; size (why not this
one?)
                   (posix-user-id) ;; uid
                   (posix-group-id) ;; gid
                   tm tm tm) ;; atime, mtime, ctime)
          ((equal? path "/hello")
           (vector (+ posix/IFREG #o440) ;; mode
                   1 ;; links
                   (string-length content) ;; size
                   (posix-user-id) ;; uid
                   (posix-group-id) ;; gid
                   tm tm tm) ;; atime, mtime, ctime)
          (else (- posix/ENOENT))))))

    :readdir (lambda (path)
      (if (equal? path "/")
          '( "." ".." "hello" )
          (- posix/ENOENT)))

    :open (lambda (path mode fd)
      (cond
        ((not (equal? path "/hello"))
         (- posix/ENOENT))
        ((not (equal? mode 0))
         (- posix/EACCESS))
        (else
         0)))

    :read (lambda (fd size offset)
      (let ((len (string-length content)))
        (if (< offset len)
            (begin
              (if (> (+ offset size) len)
                  (set! size (- len offset)))
              (substring content offset size)
              0))))))

```