Humboldt University

Computer Science Department
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Operating Systems Principles

FUSE

What is … ?

- File system
  - maps file paths (e.g., /etc/hostname) to file contents and metadata
  - Metadata includes modification times, permissions, etc.
  - File systems are ‘mounted’ over a particular directory
- Userspace
  - OS has (at least) two modes: kernel (trusted) and user
  - Kernelspace code has real ultimate power and can only be modified by root
  - Base system software like filesystems are traditionally kernel modules and not changeable by normal users

What is FUSE?

- **F**ilesystem in **USE**rspace
- Allows to implement a fully functional filesystem in a userspace program
  - Editing kernel code is not required
  - Without knowing how the kernel works
  - Usable by non privilaged users
  - Faulty implementation do not affect the system
  - More quickly/easily than traditional file systems built as a kernel module
- Not only for Linux
  - Fuse for FreeBSD
  - OSXFuse
  - Dokan (Windows)
- Wide language support: natively in C, C++, Java, C#, Haskell, TCL, Python, Perl, Shell Script, SWIG, OCaml, Pliant, Ruby, Lua, Erlang, PHP
- Low-level interface for more efficient file systems

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

FUSE Examples

- Hardware-based: ext2, iso, ZFS…
- Network-based: NFS, smb, SSH…
- Nontradtional: Gmail, MySQL…
- Loopback: compression, conversion, encryption, virus scanning, versioning…
- Synthetic: search results, application interaction, dynamic conf files…

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

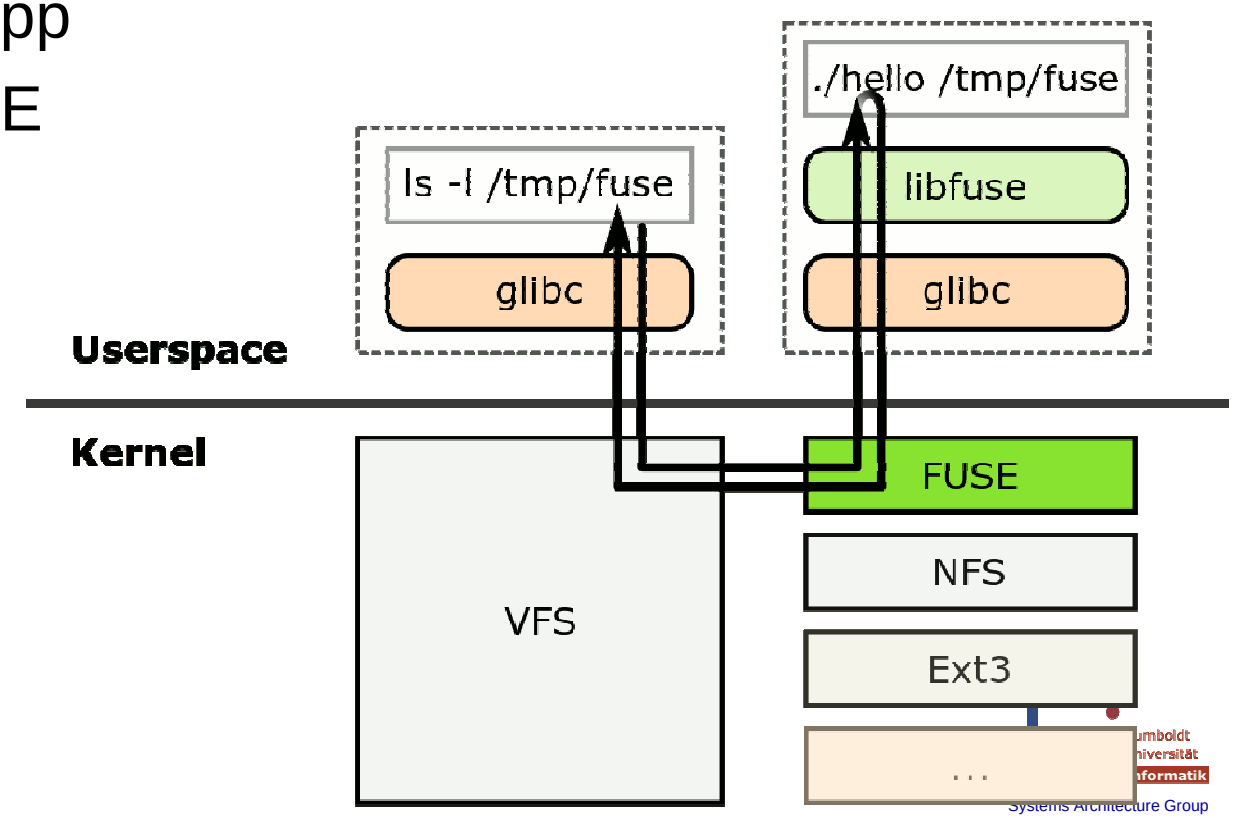
Using FUSE Filesystems

- To mount:
  - ./userspacefs ~/somedir
- To unmount:
  - fusermount -u ~/somedir
- Example sshfs
  - sshfs user@gruenau.informatik.hu-berlin.de:. /home/user/uni
  - fusermount –u /home/user/uni

How FUSE Works

- Application makes a file-related syscall
- Kernel figures out that the file is in a mounted FUSE filesystem
- The FUSE kernel module forwards the request to your userspace FUSE app
- Your app tells FUSE

  how to reply



Writing FUSE Filesystems

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Writing a FUSE Filesystem

- Write an ordinary application that defines certain functions/methods that FUSE will call to handle operations
- ~35 possible operations
- Many operations have useful defaults
  - Useful filesystems can define only ~4
  - Full-featured ones will need to define most

Defining FUSE Operations

- C: define functions and put pointers to them on a struct
- Python-fuse: operations are methods on a subclass of fuse.Fuse
- Set Fuse subclass's file_class attribute to a class that implements the file operations, or implement them on your Fuse subclass

FUSE Operations

- Directory Operations
  - readdir(path): yield directory entries for each file in the directory
  - mkdir(path, mode): create a directory
  - rmdir(path): delete an empty directory
- Metadata Operations
  - getattr(path): read metadata
  - chmod(path, mode): alter permissions
  - chown(path, uid, gid): alter ownership
- File Operations
  - mknod(path, mode, dev): create a file (or device)
  - unlink(path): delete a file
  - rename(old, new): move and/or rename a file
  - open/read/write/
- Some other stuff

Reading and Writing Files

- open(path, flags): open a file
- read(path, length, offset, fh)
- write(path, buf, offset, fh)
- truncate(path, len, fh): cut off at length
- flush(path, fh): one handle is closed
- release(path, fh): file handle is completely closed (no errors)

Operations

- Meta operations
  - fsinit(self): initialize filesystem state after being mounted (e.g. start threads)

- Other
  - statfs(path)
  - fsdestroy()
  - create(path, flags, mode)
  - utimens(path, times)
  - readlink(path)
  - symlink(target, name)
  - link(target, name)
  - fsync(path, fdatasync, fh)
  - …

FUSE Context

- struct fuse_context
  - uid: accessing user's user ID
  - gid: accessing user's group ID
  - pid: accessing process's ID
  - umask: umask of calling process
  - private_data
- struct fuse_context *fuse_get_context(void)
- Useful for nonstandard permission models and other user-specific behavior

Errors in FUSE

- Don’t have access to the user’s terminal (if any), and can only send predefined codes from the errno module
  - the error code to indicate failure
- Can log arbitrary messages to a log file for debugging

Useful Errors

- ENOSYS: Function not implemented
- EROFS: Read-only file system
- EPERM: Operation not permitted
- EACCES: Permission denied
- ENOENT: No such file or directory
- EIO: I/O error
- EEXIST: File exists
- ENOTDIR: Not a directory
- EISDIR: Is a directory
- ENOTEMPTY: Directory not empty

- errno.h, errno-base.h

fuse_lowlevel.h

- C only
- Uses numeric ‘ino’ identifiers instead of always passing full paths
- Less ‘friendly’ interface (more similar to kernel interface) allows FUSE to add less overhead

Examples

Example: hello_fs.c

- Minimal synthetic file system
- Holds a single immutable file with a pre-defined message
- Could easily be adapted to run arbitrary code to generate the file contents
- Uses 4 operations
  - readdir, open, read, getattr

General

```
#define FUSE_USE_VERSION 26

#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>

static const char *hello_str = "Hello
World!\n";
static const char *hello_path = "/hello";
```

readdir

```
static int hello_readdir(const char *path, void *buf,
                fuse_fill_dir_t filler, off_t offset,
                struct fuse_file_info *fi)
{
        (void) offset;
        (void) fi;

        if (strcmp(path, "/") != 0)
                return -ENOENT;

        filler(buf, ".", NULL, 0);
        filler(buf, "..", NULL, 0);
        filler(buf, hello_path + 1, NULL, 0);

        return 0;
}
```

open

```
static int hello_open(const char *path,
                      struct fuse_file_info *fi)
{
     if (strcmp(path, hello_path) != 0)
          return -ENOENT;

     if ((fi->flags & 3) != O_RDONLY)
          return -EACCES;

     return 0;
}
```

read

```
static int hello_read(const char *path, char *buf,
                      size_t size, off_t offset,
                        struct fuse_file_info *fi)
{
        size_t len;
        (void) fi;
        if(strcmp(path, hello_path) != 0)
                return -ENOENT;

        len = strlen(hello_str);
        if (offset < len) {
                if (offset + size > len)
                        size = len - offset;
                memcpy(buf, hello_str + offset, size);
        } else
                size = 0;

        return size;
}
```

getattr

```
static int hello_getattr(const char *path,
                         struct stat *stbuf)
{
        int res = 0;

        memset(stbuf, 0, sizeof(struct stat));
        if (strcmp(path, "/") == 0) {
                stbuf->st_mode = S_IFDIR | 0755;
                stbuf->st_nlink = 2;
        } else if (strcmp(path, hello_path) == 0) {
                stbuf->st_mode = S_IFREG | 0444;
                stbuf->st_nlink = 1;
                stbuf->st_size = strlen(hello_str);
        } else
                res = -ENOENT;

        return res;
}
```

Main

```
static struct fuse_operations hello_oper = {
      .getattr    = hello_getattr,
      .readdir    = hello_readdir,
      .open       = hello_open,
      .read       = hello_read,
};

int main(int argc, char *argv[])
{
      return fuse_main(argc, argv, &hello_oper, NULL);
}
```

Example: fusexmp.c

- Mirrors a local file hierarchy
- Simple to implement using functions in the os module
- Shows how many operations work
- Usage:

```
./fusexmp --o root=/home/ /tmp/home
```

Example: fusexmp.c

- C only
- Uses numeric ‘ino’ identifiers instead of always passing full paths
- Less ‘friendly’ interface (more similar to kernel interface) allows FUSE to add less overhead