

# FUSE

Julien Bachmann

LSE

April 21, 2006



# outline of Part1

- 1 Presentation
  - what is it?
  - using FUSE
  - what it does
- 2 Under the hood
  - the library
  - the kernel loadable module
  - communication with the VFS
  - kernel to userland

# outline of Part1

- 1 Presentation
  - what is it?
  - using FUSE
  - what it does
- 2 Under the hood
  - the library
  - the kernel loadable module
  - communication with the VFS
  - kernel to userland

# outline of Part2

- 3 Writing a LKM
  - NetBSD's LKM
  - hooks with the system
  - memory management

- 4 Problems

# outline of Part2

- 3 Writing a LKM
  - NetBSD's LKM
  - hooks with the system
  - memory management
  
- 4 Problems

## Part I

# FUSE for Linux

# What is it?

- FUSE is Linux a package that allow you to implement a fully functional filesystem in userspace program
- Compose of two parts: a loadable kernel module and a library to include in userspace programs

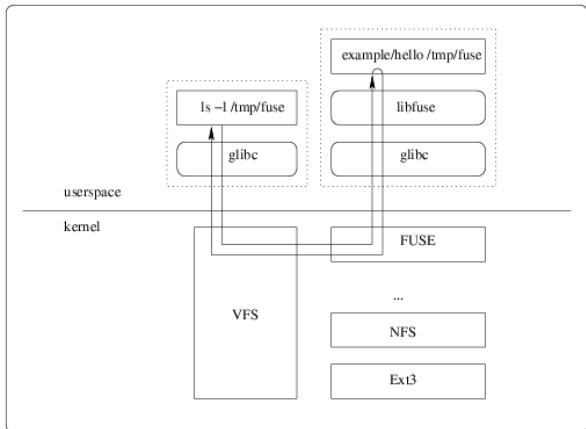
# Using FUSE

- include fuse.h
- fill fuse\_operation structure (all methods are optional but some are essential)
- call fuse\_main()

# Example

```
static int hello_open(const char *path,
                     struct fuse_file_info *fi)
{...}
static int hello_getattr(const char *path,
                        struct stat *stbuf)
{...}
static struct fuse_operations hello_oper = {
    .getattr      = hello_getattr,
    .open        = hello_open,
};
int main(int argc, char *argv[])
{
    return fuse_main(argc, argv, &hello_oper);
}
```

# what it does



# mounting the filesystem

## fuse\_mount

- creation of a socketpair stored in the environment
- fork, to execute fusemount which has root execution right

## child

- seteuid setegid to restore root uid and gid
- open /dev/fuse and send file-descriptor back to his parent using credentials

## fuse\_mount

- call mount()
- wait messages from lkm

# mounting the filesystem

## fuse\_mount

- creation of a socketpair stored in the environment
- fork, to execute fusemount which has root execution right

## child

- seteuid setegid to restore root uid and gid
- open /dev/fuse and send file-descriptor back to his parent using credentials

## fuse\_mount

- call mount()
- wait messages from lkm

## mounting the filesystem

### fuse\_mount

- creation of a socketpair stored in the environment
- fork, to execute fusemount which has root execution right

### child

- seteuid setegid to restore root uid and gid
- open /dev/fuse and send file-descriptor back to his parent using credentials

### fuse\_mount

- call mount()
- wait messages from lkm

# mounting the filesystem

## fuse\_mount

- creation of a socketpair stored in the environment
- fork, to execute fusemount which has root execution right

## child

- seteuid setegid to restore root uid and gid
- open /dev/fuse and send file-descriptor back to his parent using credentials

## fuse\_mount

- call mount()
- wait messages from lkm

## mounting the filesystem

### fuse\_mount

- creation of a socketpair stored in the environment
- fork, to execute fusemount which has root execution right

### child

- seteuid setegid to restore root uid and gid
- open /dev/fuse and send file-descriptor back to his parent using credentials

### fuse\_mount

- call mount()
- wait messages from lkm

## mounting the filesystem

### fuse\_mount

- creation of a socketpair stored in the environment
- fork, to execute fusemount which has root execution right

### child

- seteuid setegid to restore root uid and gid
- open /dev/fuse and send file-descriptor back to his parent using credentials

### fuse\_mount

- call mount()
- wait messages from lkm

## the lkm is composed of two main parts:

- VFS interaction
- userland interaction

## the lkm is composed of two main parts:

- VFS interaction
- userland interaction

# lkm initialisation

```
static int __init fuse_init(void)
{
    //FIXME
}
static int __init fuse_exit(void)
{
    //FIXME
}
module_init(fuse_init);
module_exit(fuse_exit);
```

# vfs hooks

- fill VFS operations structure
- call `register_filesystem()`

# vfs hooks

- fill VFS operations structure
- call `register_filesystem()`

# communication with the VFS

The VFS will:

- find the inode
- call the function that FUSE put in the inode when it allocated it

# communication with the VFS

The VFS will:

- find the inode
- call the function that FUSE put in the inode when it allocated it

## sending a request to userspace

- FUSE is registered as a device driver too
- he will fill the buffer given by the userland when it called `read()`

## sending a request to userspace

- FUSE is registered as a device driver too
- he will fill the buffer given by the userland when it called `read()`

## processing a request in userspace

- the library reads data on `/dev/fuse`'s filehandle
- check if the action is implemented
- call the action

## feedback to the lkm

The inode hash table is modified if the action was successful

## Part II

# NetBSD portage

# NetBSD's LKM

## Several kind of lkm

- system call modules
- virtual file system modules
- device driver module
- ...

# NetBSD's LKM

Several kind of lkm

- system call modules
- virtual file system modules
- device driver module
- ...

## virtual file system module

As in Linux, you have to fill a structure with callback for VFS operations

# virtual file system module

## declaration

```
MOD_VFS("fusefs", -1, &fusefs_vfsops);
```

## entry point

```
int      fusefs_lkmentry(struct lkm_table* lkmtpl,
                        int          cmd,
                        int          ver)
{
    DISPATCH(lkmtpl, cmd, ver, load, unload,
             lkm_nofunc)
}
```

## allocating inodes

Can't use malloc because we are in the kernel

- in Linux, you need to use a kernel cache
- in NetBSD, simply use `getnewvnode()` and `kmalloc()`

## allocating inodes

Can't use malloc because we are in the kernel

- in Linux, you need to use a kernel cache
- in NetBSD, simply use `getnewvnode()` and `kmalloc()`

# problems

- translation from Linux to NetBSD
- complex code