

# Sistemas Operativos

Universidad Complutense de Madrid

## Práctica 2

*Gestión de un sistema de ficheros en espacio de usuario*

J.C. Sáez

# Objetivos

## Objetivos

- Comprender las llamadas al sistema y funciones en GNU/Linux para manejo de ficheros y directorios
- Familiarizarse con la problemática asociada a la gestión de un sistema de ficheros
- Entender los fundamentos de FUSE

# FUSE

## FUSE: *Filesystem in USErspace*

- Framework para implementar **sistemas de ficheros en espacio de usuario**
- Implementación disponible para Linux, FreeBSD, Android, OS X,...

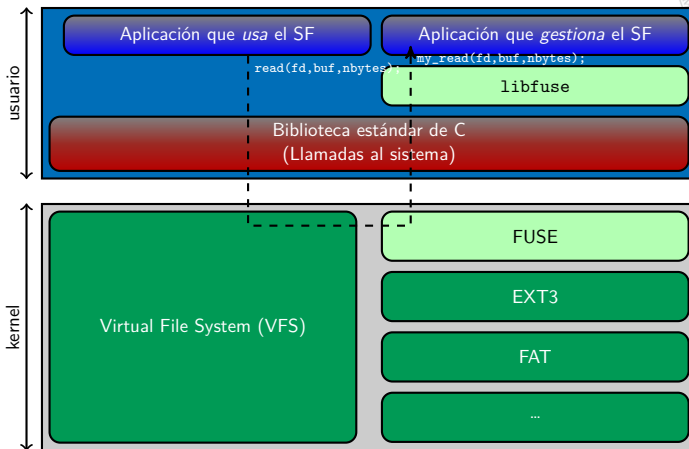
## Idea general

- Programa de usuario actúa como *gestor* del sistema de ficheros
  - Se comporta como un servidor que responde a peticiones sobre el sistema de ficheros
  - Implementa subconjunto de llamadas al sistema (LLSS) sobre ficheros
- Cuando cualquier proceso accede a un fichero de este sistema de ficheros FUSE invoca las funciones código del gestor que implementan las LLSS sobre el sistema de ficheros

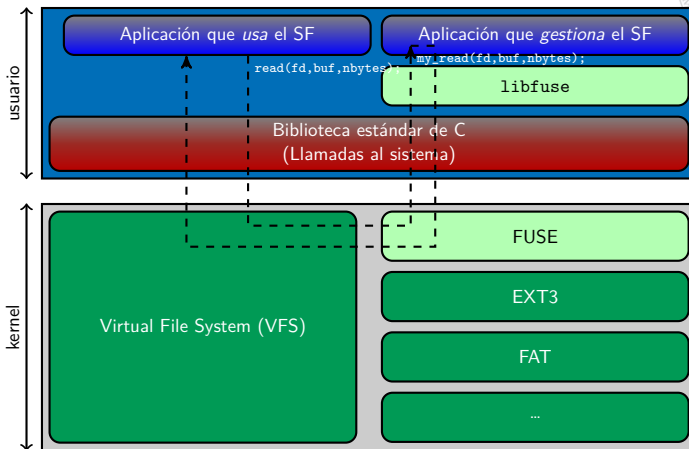
# FUSE: Visión Global



# FUSE: Visión Global



# FUSE: Visión Global



# Interfaz de operaciones sobre ficheros

## Interfaz de Operaciones (/usr/include/fuse/fuse.h)

```
struct fuse_operations {
    int (*getattr) (const char *, struct stat *);
    int (*mknod) (const char *, mode_t, dev_t);
    int (*mkdir) (const char *, mode_t);
    int (*unlink) (const char *);
    int (*rmdir) (const char *);
    int (*symlink) (const char *, const char *);
    int (*rename) (const char *, const char *);
    int (*link) (const char *, const char *);
    int (*chmod) (const char *, mode_t);
    int (*chown) (const char *, uid_t, gid_t);
    int (*truncate) (const char *, off_t);
    int (*open) (const char *, struct fuse_file_info *);
    int (*read) (const char *, char *, size_t, off_t, struct
        fuse_file_info *);
    int (*write) (const char *, const char *, size_t, off_t,
        struct fuse_file_info *);
    int (*release) (const char *, struct fuse_file_info *);
    int (*opendir) (const char *, struct fuse_file_info *);
    int (*readdir) (const char *, void *, fuse_fill_dir_t, off_t,
        struct fuse_file_info *);
    int (*releasedir) (const char *, struct fuse_file_info *);
    ...
};
```

## Operaciones básicas (I)

```
int (*getattr)(const char *, struct stat *);
```

- Función llamada cuando se quieren obtener los atributos de un fichero (ej: `$ stat fichero`). El primer parámetro indica la ruta del fichero. El segundo parámetro es la estructura `stat` a rellenar.



## Operaciones básicas (II)

```
int (*open)(const char *, struct fuse_file_info *);
```

- Se llama al abrir un fichero.
- La función comprueba si se puede abrir el fichero, en caso afirmativo se devuelve cero.
- El primer parámetro es la ruta del fichero.
- El segundo parámetro es una estructura que describe al fichero abierto en FUSE.
  - Campos más relevantes de la estructura:
    - flags: almacena los flags de apertura (solo lectura, solo escritura, lectura-escritura)
    - fh: En la práctica este campo se usa para guardar el número de nodo-i del fichero abierto.

## Operaciones básicas (III)

```
int (*read)(const char *, char *, size_t, off_t, struct
fuse_file_info *);
```

- Se llama cuando un proceso lee bytes de un fichero
  - Primer parámetro: ruta del fichero
  - Segundo parámetro: buffer donde almacenar los bytes leídos del fichero
  - Tercer parámetro: número de bytes a leer
  - Cuarto parámetro: ubicación del puntero de posición del fichero abierto
  - Quinto parámetro: estructura que representa al fichero abierto en FUSE
- La función debe devolver el número de bytes leídos

## Operaciones básicas (IV)

```
int (*readdir)(const char *, void *, fuse_fill_dir_t, off_t  
    , struct fuse_file_info *);
```

- Se utiliza para leer un directorio
  - Primer parámetro: ruta del directorio
  - Segundo parámetro: estructura que hay que rellenar para devolver los datos
  - Tercer parámetro: función usada para rellenar la estructura pasada como segundo
  - Los dos últimos parámetros se pueden ignorar para ejemplos sencillos.

## Operaciones básicas (V)

### Valor de retorno de las operaciones (`fuse_operations`)

- Las funciones deben devolver 0 en caso de éxito y un número negativo indicando el error en caso de fallo.
- **Excepción:** Las llamadas a `read/write` deben devolver un número positivo indicando los bytes leídos/escritos, 0 en caso de EOF o número negativo en caso de error.

### Para más información

- API de FUSE:
  - <http://libfuse.github.io/doxygen/index.html>
- Documentación general:
  - *Develop your own filesystem with FUSE*
  - *FUSE at LWN.net*

# Gestión de un SF con FUSE

## ¿Cómo se usa?

- Crear programa de usuario que gestiona el SF
  - Hay que incluir `fuse.h` y enlazar con `libfuse`
- El programa ha de instanciar la interfaz de operaciones:
  - 1 Definir variable tipo `struct fuse_operations`
    - Establece la asociación entre las operaciones de la interfaz y las funciones del programa que se invocan
    - Cada función → Mismo prototipo que operación correspondiente
    - No es necesario definir todas las operaciones
  - 2 Para “conectar” con FUSE el programa debe invocar `fuse_main()`
    - La función acepta como parámetro (entre otros) un puntero a la variable que instancia la interfaz
    - `fuse_main()` es bloqueante
    - Al invocarla, el programa se queda a la espera de peticiones

# Hello FUSE

## Ejemplo (`hello_fuse.c`)

- Sistema de ficheros de solo lectura “almacenado” en memoria
- Contiene un único fichero llamado `hello`
  - Almacena la cadena “Hello World!”

## Ejemplo (1/5)

### hello\_fuse.c

```
#include <fuse.h>
#include <stdio.h>
#include <string.h>

static int hello_getattr(const char *path, struct stat *stbuf);
static int hello_readdir(const char *path, void *buf,
    fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi);
static int hello_open(const char *path, struct fuse_file_info *fi);
static int hello_read(const char *path, char *buf, size_t size,
    off_t offset, struct fuse_file_info *fi);

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);
}
```

## Ejemplo (2/5)

### hello\_fuse.c

```
/* Ruta FUSE del único fichero del sistema. Las rutas siempre son
   'absolutas' para el programa de usuario gestor. */
static const char *hello_path = "/hello";

/* Contents of file "hello" */
static const char *hello_str = "Hello World!\n";

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;
}
```



## Ejemplo (3/5)

### hello\_fuse.c

```
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;
}
```

## Ejemplo (4/5)

### hello\_fuse.c

```
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;
}
```

## Ejemplo (5/5)

### hello\_fuse.c

```
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;
}
```

# Ejemplo de ejecución

terminal 1

```
$ gcc -Wall hello_fuse.c `pkg-config fuse --cflags --libs` -o hello_fuse
$ mkdir /tmp/fuse
$ ./hello_fuse /tmp/fuse -d
FUSE library version: 2.9.0
nullpath_ok: 0
nopath: 0
utime_omit_ok: 0
unique: 1, opcode: INIT (26), nodeid: 0, insize: 56, pid: 0
INIT: 7.22
flags=0x0000f7fb
max_readahead=0x00020000
INIT: 7.19
flags=0x00000011
max_readahead=0x00020000
max_write=0x00020000
max_background=0
congestion_threshold=0
unique: 1, success, outsize: 40
unique: 2, opcode: LOOKUP (1), nodeid: 1, insize: 47, pid: 2645
...
```

## Ejemplo de ejecución (cont.)

- Mientras tanto, en otra terminal...

terminal 2

```
$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,relatime,size=10240k,nr_inodes=254708,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmx=0)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=204912k,mode=755)
/dev/disk/by-uuid/4686626b-cfc7-41f2-b295-cb609d68b1a5 on / type ext4 (rw,relatime)
....
hello_fuse on /tmp/fuse type fuse.hello_fuse (rw,nosuid,nodev,relatime,user_xattr)
$ ls -l /tmp/fuse
total 0
-r--r--r--  1 root root 13 Jan  1  1970 hello
$ cat /tmp/fuse/hello
Hello World!
$ echo Heyyy > /tmp/fuse/a.txt
bash: /tmp/fuse/a.txt: Function not implemented
$ fusermount -u /tmp/fuse
```

## Práctica propuesta (I)

- En esta práctica se llevará a cabo la implementación de un sistema de ficheros sencillo tipo UNIX
- El sistema de ficheros residirá en un único archivo (*disco virtual*) y se gestionará mediante un programa de usuario
  - Se proporciona programa gestor basado en FUSE
  - Práctica: extender la funcionalidad del gestor del sistema de ficheros

## Práctica propuesta (II)

### Características del sistema de ficheros

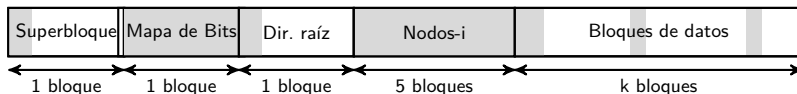
- 1 Organización de la partición muy similar a la del SF tipo UNIX tradicional pero sin sección *Boot*
- 2 Incluye un único directorio en el sistema (raíz) que puede almacenar hasta 100 entradas
  - El directorio raíz reside en una región específica del *disco virtual*
- 3 Cada fichero está representado internamente mediante un nodo-*i*, que incluye, entre otras cosas, 100 índices directos a bloques de datos
  - El tamaño máximo de cada fichero es, por tanto,  $100 \cdot \text{TamBloque}$
  - No se hace uso de índices indirectos
  - Los bloques de datos del fichero no han de ser necesariamente contiguos

# Implementación (I)

## ■ Representación en memoria del sistema de ficheros

```
typedef struct MyFileSystemStructure {
    int fdVirtualDisk;           /* Descriptor de fichero del disco virtual */
    SuperBlockStruct superBlock; /* Superbloque */
    BIT bitMap[NUM_BITS];        /* Mapa de bits */
    DirectoryStruct directory;   /* Directorio raíz */
    NodeStruct* nodes[MAX_NODES]; /* Array de punt. a Nodos-i */
    int numFreeNodes;            /* Número de nodos-i libres */
} MyFileSystem;
```

## ■ Estructura del sistema de ficheros en disco (contenido del disco virtual)





## Implementación (II)

- El tamaño de bloque está definido por la macro `BLOCK_SIZE_BYTES` en `common.h`
  - Valor por defecto 4KB
- Los bloques de datos de los ficheros no se almacenan en memoria, sólo en disco
  - Por este motivo la estructura `MyFileSystemStructure` no tiene campo para los datos
- El mapa de bits se implementa como un array de enteros
  - 0 → libre, 1 → ocupado
  - Los 8 primeros bloques de disco (superbloque, mapa de bits, dir. Raíz, y 5 bloques para nodos-i) se han de marcar como ocupados al formatear el disco virtual
  - El mapa de bits sólo lleva la cuenta de bloques libres y ocupados

## Implementación (III)

- Las representaciones del vector de nodos- $i$  en memoria y en disco difieren:
  - *Disco*: Se almacenan todos los nodos- $i$ , se usen o no.
    - Si el nodo- $i$  está asignado a un fichero, su campo `freeNode` valdrá 0.
    - Si está libre, dicho campo almacenará un 1.
  - *Memoria*: Se mantiene un array de punteros a `NodeStructure`.
    - Si el nodo- $i$   $k$ -ésimo no está asignado a ningún fichero, el array de punteros almacenará `NULL` en la posición  $k$
    - En caso contrario, apuntará a una estructura `NodeStructure` válida.

```
typedef struct NodeStructure {  
    int numBlocks;           // Núm. bloques  
    int fileSize;            // Tamaño fichero  
    time_t modificationTime; // Tiempo de modificación  
    DISK_LBA blocks[MAX_BLOCKS_PER_FILE]; // Bloques  
    BOOLEAN freeNode;        // Nodo- $i$  libre  
} NodeStruct;
```

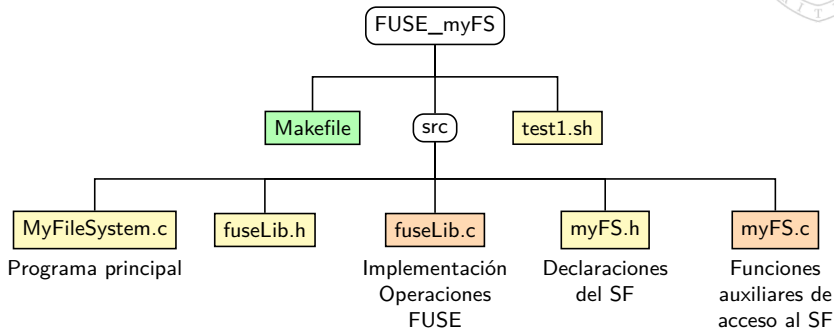
## Implementación (IV)

- El directorio raíz se implementa como una tabla (array) con un contador de ficheros asociado
  - Cada entrada de la tabla almacena el nombre del fichero, su número de nodo-i, y un indicador de si la entrada del directorio está en uso o no

```
typedef struct FileStructure {
    int nodeId; // Associated inode
    char fileName[MAX_LEN_FILE_NAME + 1]; // File name
    BOOLEAN freeFile; // Free file
} FileStruct;

typedef struct DirectoryStructure {
    int numFiles; // Num files
    FileStruct files[MAX_FILES_PER_DIRECTORY]; // Directory entries
} DirectoryStruct;
```

# Estructura del proyecto C



## Desarrollo de la práctica

- 1 Implementar las operaciones `read` y `unlink` del sistema de ficheros:
  - Añadir funciones `my_read()` y `my_unlink()` en `fuseLib.c`
  - Realizar modificaciones pertinentes en la interfaz de operaciones (estructura `fuse_operations`)
  - Consultar implementación del resto de operaciones que se proporcionan
- 2 Implementar *script* de comprobación (`test2.sh`)
  - Más información en el guión

## Algunas aclaraciones

### Advertencias

- Aunque sólo haya que modificar `fuseLib.c`, es necesario analizar el código del resto del proyecto para comprender su funcionamiento
  - En la parte extra de la práctica se podrían pedir modificaciones del resto del código
- En esta práctica pueden usarse las llamadas al sistema de Linux (`open()`, `read()`, `write()`, `close()`, ...) para el acceso al disco virtual desde el programa, pero NO las de la biblioteca estándar (`fopen()`, `fread()`, `fwrite()`, `fclose()`, ...)

## Parte opcional

- La parte básica de la práctica tiene una **limitación**: *el gestor del sistema de ficheros NO trabaja con discos virtuales ya formateados*
  - Siempre se formatea el disco al arrancar el gestor

### Parte opcional propuesta

- Modificar el programa gestor para que sea posible trabajar con discos virtuales formateados y (posiblemente) con contenido
- Para ello el programa se invocará de la siguiente forma
 

```
./fs-fuse -a <virtual-disk> -m -f '-d -s <mount-point>
```

  - El procesamiento asociado a la línea de comando ya está implementado
- Esta parte opcional requiere modificar el fichero `myFs.c`

## Parte opcional (II)

- Se proporciona el código completo de la función `myMount()` (en `myFs.c`), pero no así el de las funciones auxiliares que ésta invoca:

- `int readBitMap(MyFileSystem* myFileSystem);`
- `int readDirectory(MyFileSystem* myFileSystem);`
- `int readSuperblock(MyFileSystem* myFileSystem);`
- `int readInodes(MyFileSystem* myFileSystem);`

- La implementación de estas funciones auxiliares ha de completarse en `myFs.c`
  - **Pista:** analizar el código de las funciones `update***()` correspondientes



## Parte opcional: Ejemplo de ejecución

terminal 1

```
jcsaez@debian:~/FUSE_myFS$ ./fs-fuse -m -a virtual-disk -f '-d -s mount-point'
SF: virtual-disk, 2097152 B (4096 B/block), 512 blocks
1 block for SUPERBLOCK (32 B)
1 block for BITMAP, covering 1024 blocks, 4194304 B
1 block for DIRECTORY (2404 B)
5 blocks for inodes (424 B/inode, 45 inodes)
501 blocks for data (2052096 B)
Volume mounted successfully!
File system available
FUSE library version: 2.9.0
nullpath_ok: 0
nopath: 0
utime_omit_ok: 0
unique: 1, opcode: INIT (26), nodeid: 0, insize: 56, pid: 0
INIT: 7.22
flags=0x0000f7fb
max_readahead=0x00020000
  INIT: 7.18
  flags=0x00000011
  max_readahead=0x00020000
  max_write=0x00020000
  max_background=0
  congestion_threshold=0
  unique: 1, success, outsize: 40
...
```

## Parte opcional: Ejemplo de ejecución

- En otra terminal, comprobamos que los ficheros creados previamente en el disco virtual están disponibles

terminal 2

```
jcsaez@debian:~/FUSE_myFS$ ls mount-point/  
a.txt  b.txt  test.sh  
jcsaez@debian:~/FUSE_myFS$ cd mount-point/  
jcsaez@debian:~/FUSE_myFS/mount-point$ cat a.txt  
Hello  
jcsaez@debian:~/FUSE_myFS$
```