Sistemas Operativos

Curso

2014-2015

Práctica 2

Implementación de un sistema de ficheros sencillo

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

Contenido

FUSE

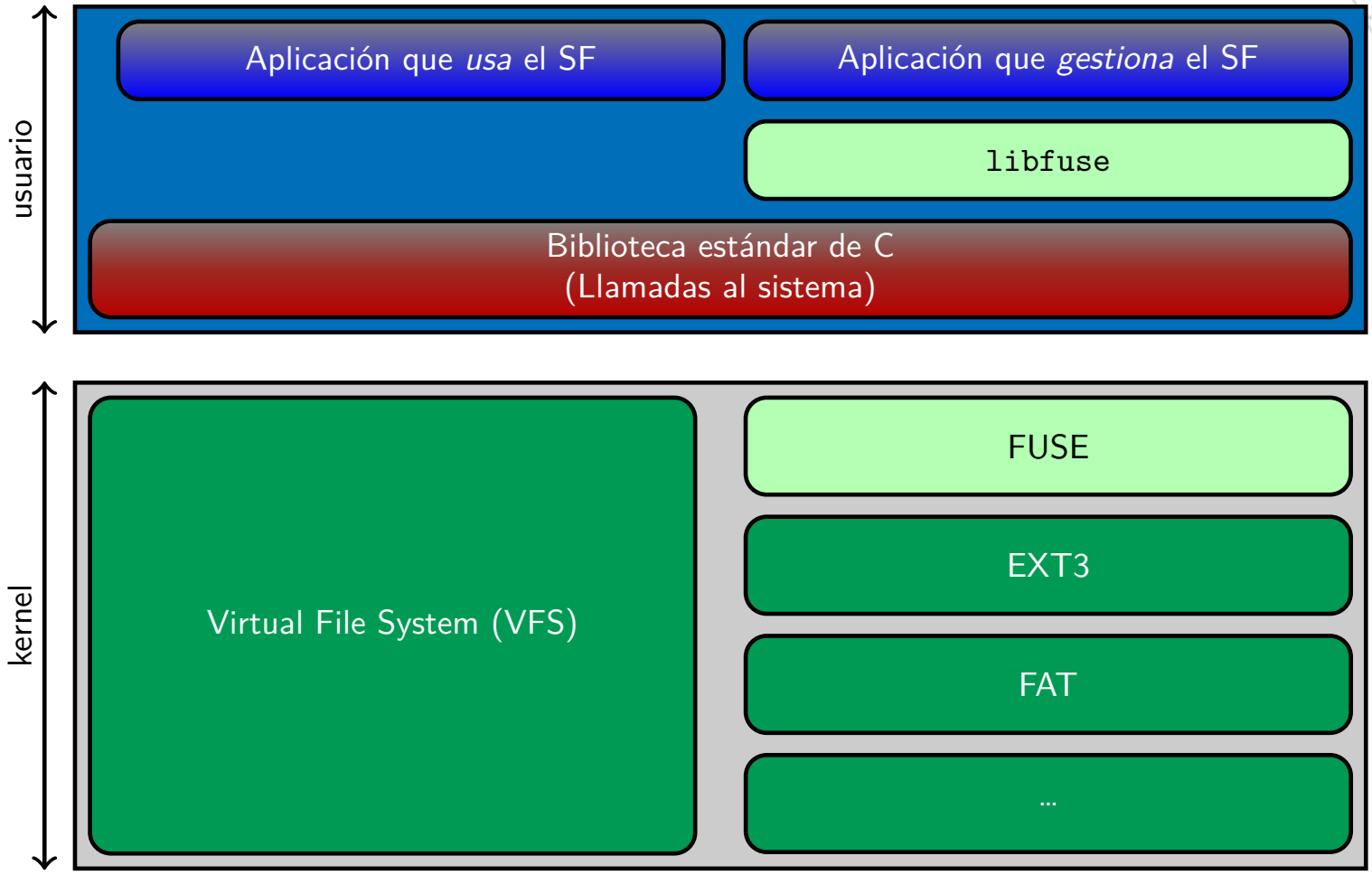
FUSE: *Filesystem in USErspace*

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

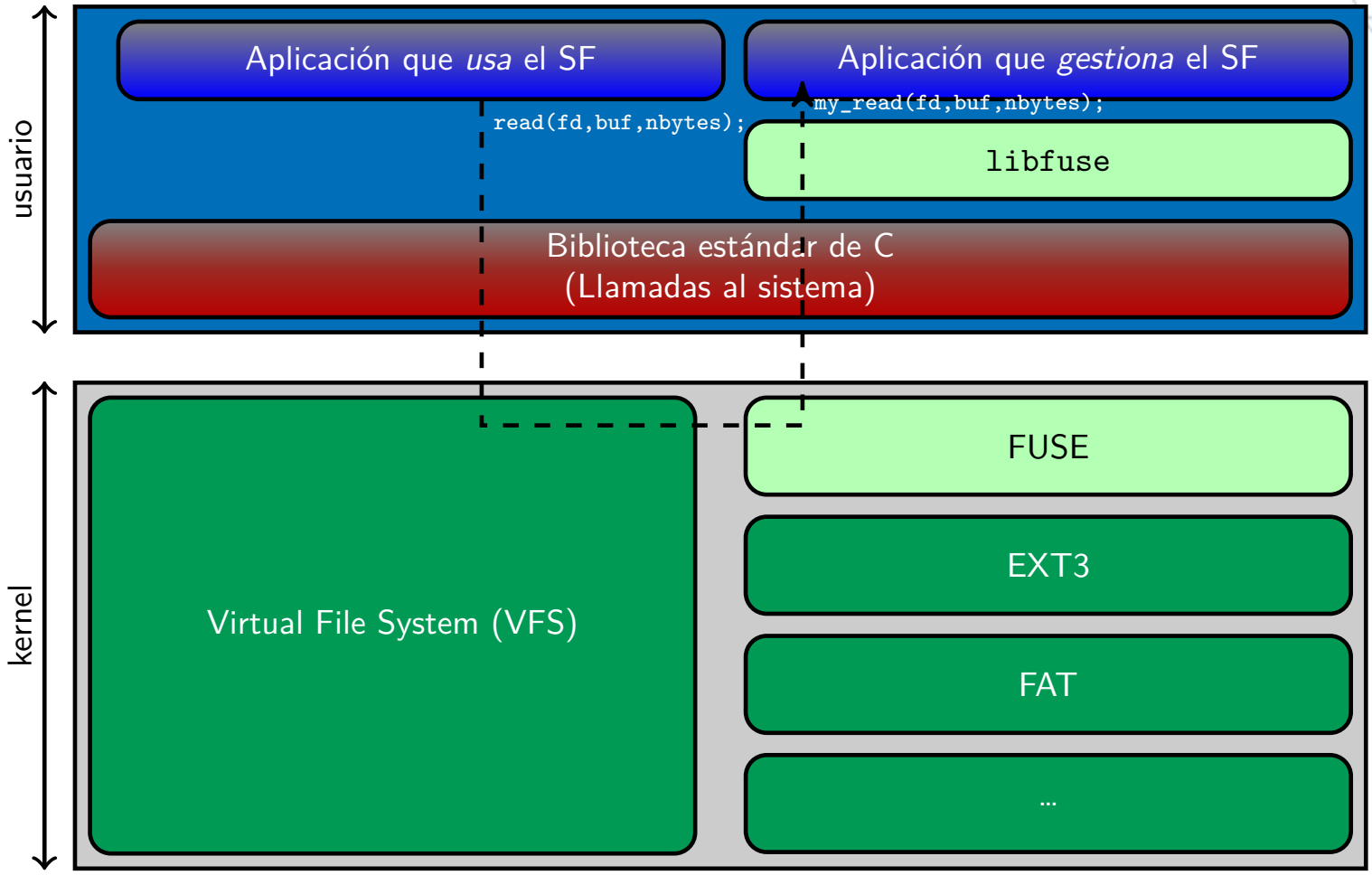
Modo de uso

- Un 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

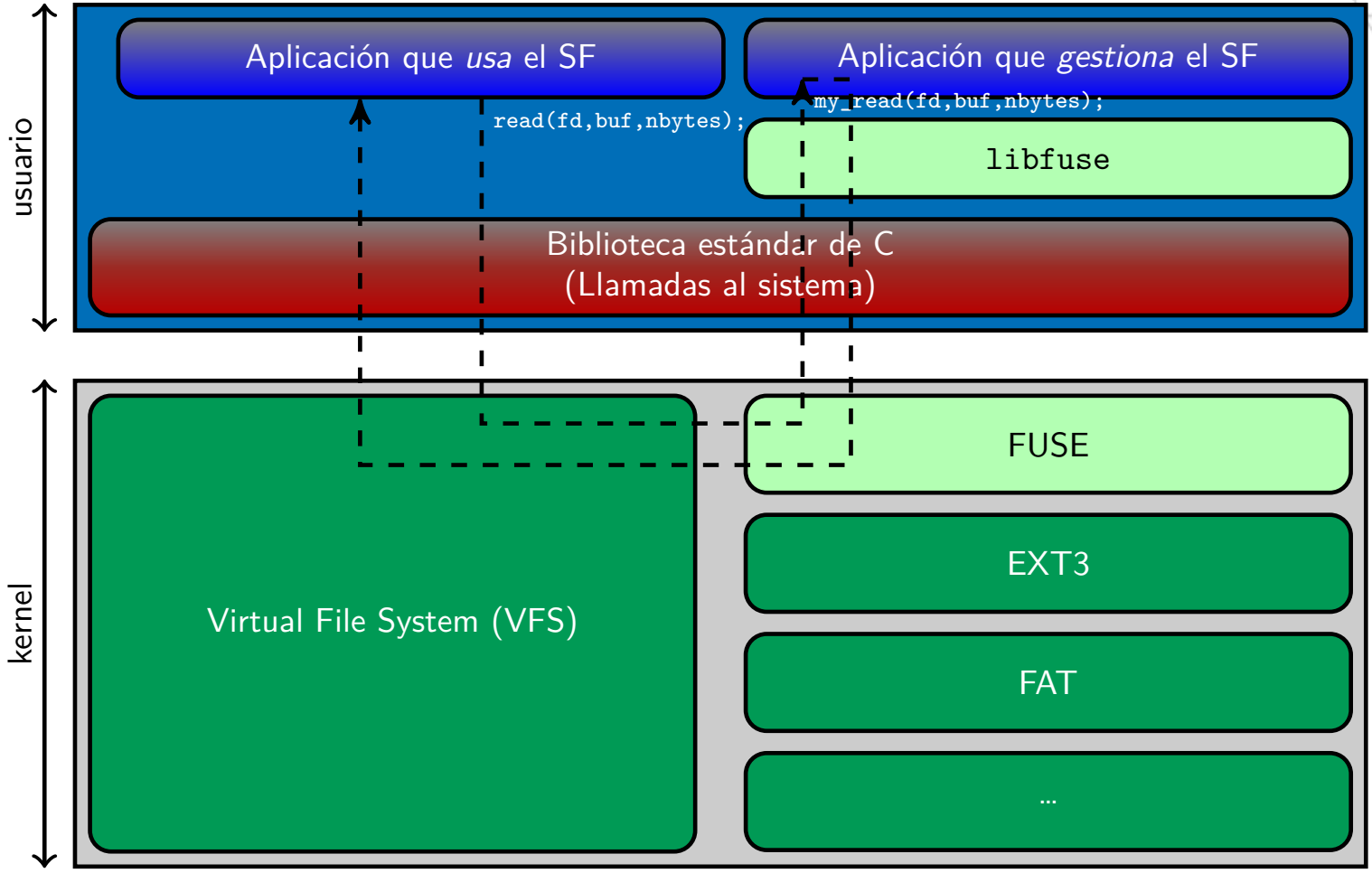
FUSE: Visión Global



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 *);
    ...
};
```

Descripción operaciones (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.

```
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.
  - Entre otras cosas, contiene información acerca de los flags de apertura.
  - En la práctica, usaremos el campo `fh` de esta estructura para guardar el número de nodo-i del fichero abierto.

Descripción operaciones (II)

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

- Se llama al leer un fichero. El primer parámetro es la ruta del fichero, el segundo es el buffer donde almacenar los datos, el tercer parámetro es la cantidad de bytes a leer, el cuarto el desplazamiento (ubicación del puntero de posición del fichero abierto) y el quinto es el mismo que el de open. Se devuelve la cantidad de bytes leídos.

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

- Se utiliza para leer un directorio. El primer parámetro es la ruta del directorio, el segundo una estructura que hay que rellenar, el tercero es una función usada para rellenar la estructura del segundo parámetro y los otros dos se pueden ignorar para ejemplos sencillos.

Descripción operaciones (III)

Para más información...

- API de FUSE:
  - `http://fuse.sourceforge.net/doxygen/index.html`
- Documentación general:
  - `http://fuse.sourceforge.net/`

Valor de retorno de las operaciones

- 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 deden devolver un número positivo indicando los bytes leídos/escritos, 0 en caso de EOF o número negativo en caso de error.

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";

/* Contenido del fichero "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

```
jcs@debian:~$ gcc -Wall hello_fuse.c `pkg-config fuse --cflags --libs` -o hello_fuse
jcs@debian:~$ mkdir /tmp/fuse
jcs@debian:~$  ./hello_fuse /tmp/fuse -d
FUSE library version: 2.9.3
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

```
jcs@debian:~$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,relatime)
udev on /dev type devtmpfs (rw,relatime,size=10240k,nr_inodes=3084386,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,relatime,size=4937580k,mode=755)
/dev/sda1 on / type ext3 (rw,relatime,errors=remount-ro,data=ordered)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
...
hello_fuse on /tmp/fuse type fuse.hello_fuse (rw,nosuid,nodev,relatime,user_id=1001,gr
jcs@debian:~$ ls -l /tmp/fuse
total 0
-r--r--r--  1 root root 13 Jan  1  1970 hello
jcs@debian:~$ cat /tmp/fuse/hello
Hello World!
jcs@debian:~$ echo Heyyy > /tmp/fuse/a.txt
bash: /tmp/fuse/a.txt: Función no implementada
jcs@debian:~$ fusermount -u /tmp/fuse
```

Contenido

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
  - Programa gestor utiliza libfuse

Práctica propuesta (II)

Características del sistema de ficheros de la práctica

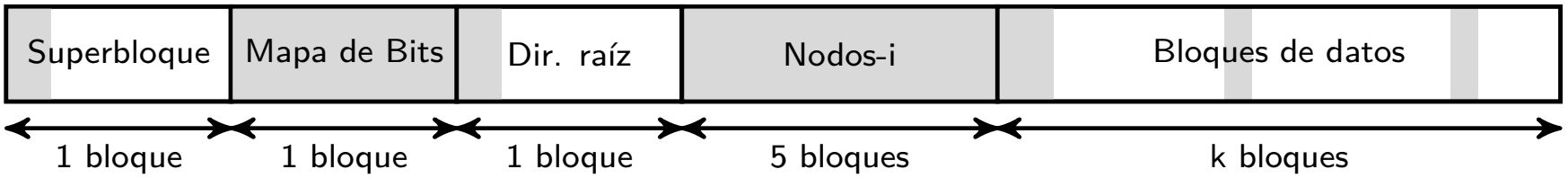
1. Organización de la partición muy similar a la del SF tipo UNIX tradicional pero sin sección *Boot*
2. Incluye un único directorio en el sistema (raíz) que puede almacenar hasta 100 entradas
   - El directorio raíz reside en una región específica del *disco virtual*
3. Cada fichero está representado 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*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 MiSistemaDeFicheros {
    int fdDiscoVirtual;     /* Descriptor de fichero del disco virtual */
    EstructuraSuperBloque superBloque; /* Superbloque */
    BIT mapaDeBits[NUM_BITS];          /* Mapa de bits */
    EstructuraDirectorio directorio; /* Directorio raíz */
    EstructuraNodoI* nodosI[MAX_NODOSI]; /* Array de punt. a Nodos-i */
    int numNodosLibres;                /* Número de nodos-i libres */
} MiSistemaDeFicheros;
```

- Estructura del sistema de ficheros en disco (contenido del disco virtual)



Implementación (II)

- El tamaño de bloque está definido por la macro `TAM_BLOQUE_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 `MiSistemaDeFicheros` 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. Raiz, 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:
  - *Memoria*: Se mantiene un array de punteros a `EstructuraNodoI`.
    - 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 `EstructuraNodoI` válida.
  - *Disco*: Se almacenan todos los nodos-i, se usen o no.
    - Si el nodo-i está asignado a un fichero, su campo `libre` valdra 0.
    - Si está libre, dicho campo almacenará un 1.

```
typedef struct EstructuraNodoI {
  int numBloques;                              // Núm. bloques
  int tamArchivo;                              // Tamaño archivo
  time_t tiempoModificado;                     // Tiempo de modificación
  DISK_LBA idxBloques[MAX_BLOQUES_POR_ARCHIVO]; // Dir. bloques
  BOOLEAN libre;                               // Nodo libre
} EstructuraNodoI;
```

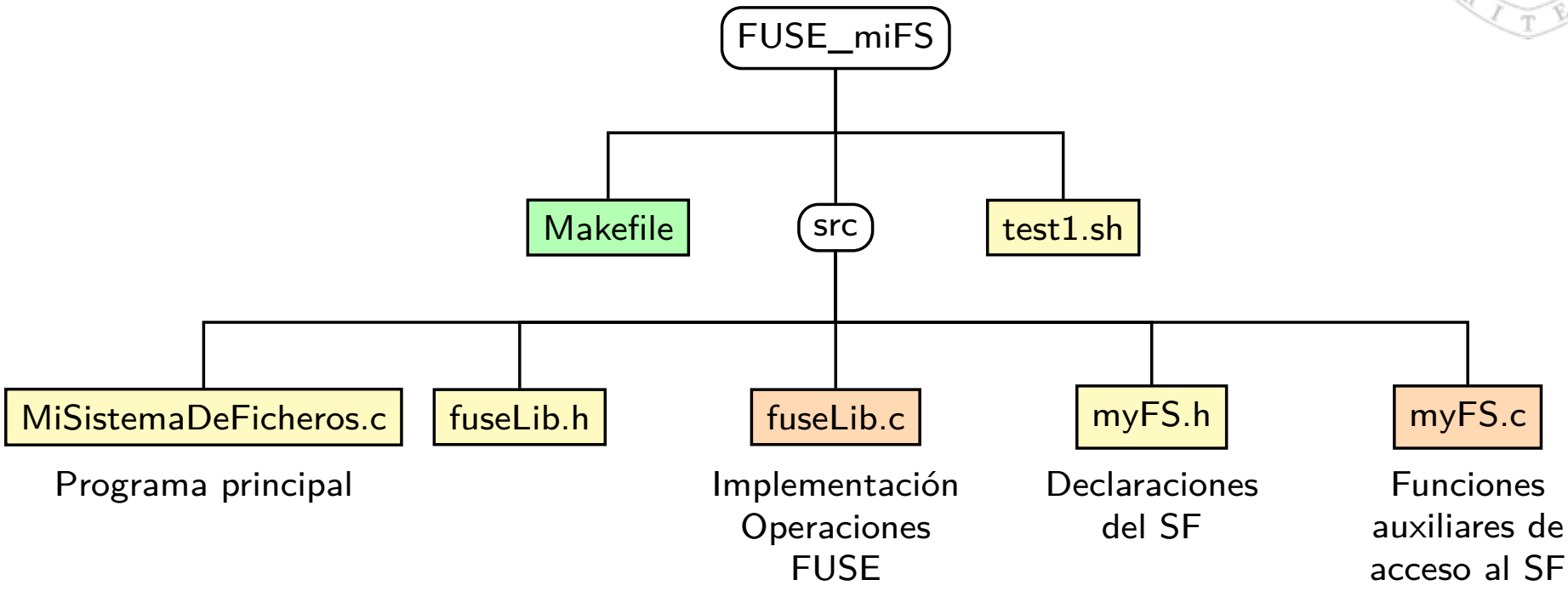
Implementación (IV)

- 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 EstructuraArchivo {
  int idxNodoI;                              // Núm. nodo-i asociado
  char nombreArchivo[MAX_TAM_NOMBRE_ARCHIVO+1]; // Nombre archivo
  BOOLEAN libre;                             // Entrada libre
} EstructuraArchivo;

typedef struct EstructuraDirectorio {
  int numArchivos;                                    // Núm. entradas directorio
  EstructuraArchivo archivos[MAX_ARCHIVOS_POR_DIRECTORIO]; // Tabla de entradas
} EstructuraDirectorio;
```

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

```
./sf-fuse -a <disco-virtual> -m -f '-d -s <punto-montaje>'
```

  - 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 leeMapaDebits(MiSistemaDeFicheros* miSistemaDeFicheros);`
  - `int leeDirectorio(MiSistemaDeFicheros* miSistemaDeFicheros);`
  - `int leeSuperbloque(MiSistemaDeFicheros* miSistemaDeFicheros);`
  - `int leeNodosI(MiSistemaDeFicheros* miSistemaDeFicheros);`
- La implementación de estas funciones auxiliares ha de completarse en `myFs.c`
  - **Pista**: analizar el código de las funciones `escribe***()` correspondientes

Parte opcional: Ejemplo de ejecución

terminal 1

```
jcsaez@debian:~/FUSE_miFS$ ./sf-fuse -a disco -m -f '-d -s punto-montaje'
SF: disco, 2097152 B (4096 B/bloque), 512 bloques
1 bloque para SUPERBLOQUE (32 B)
1 bloque para MAPA DE BITS, que cubre 1024 bloques, 4194304 B
1 bloque para DIRECTORIO (2404 B)
5 bloques para nodos-i (a 424 B/nodo-i, 45 nodos-i)
501 bloques para datos (2052096 B)
Carga completada!
Sistema de ficheros disponible
FUSE library version: 2.9.3
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
...
```

Parte opcional: Ejemplo de ejecución (cont.)

- En otra terminal, comprobamos que los datos del disco virtual están disponibles.

terminal 2

```
jcsaez@debian:~/FUSE_miFS$ ls punto-montaje/
a.txt  fichero1.txt  fichero2.txt
jcsaez@debian:~/FUSE_miFS$ cat a.txt
Hola
jcsaez@debian:~/FUSE_miFS$
```

Entrega de la práctica

- Hasta el 24 de noviembre a las 8:55h
- Para realizar la entrega de cada práctica de la asignatura debe subirse un único fichero ".zip" o ".tar.gz" al Campus Virtual
  - Ha de contener todos los ficheros necesarios para compilar la práctica (fuentes + Makefile).
  - Debe ejecutarse "make clean" antes de generar el fichero comprimido
  - Nombre del fichero comprimido:

    `L<num_laboratorio>_P<num_puesto>_Pr<num_práctica>.tar.gz`

Estructura entrega (en fichero .zip o .tar.gz)