

# Sistemas Operativos



Curso  
2017-2018

Módulo 2: Gestión de Ficheros  
Práctica 2: Sistema de Ficheros

# Agenda



## 1 Objetivo

## 2 Proyecto FUSE\_myFS

## 3 myFS

## 4 FUSE y fuseLib

## 5 Parte Obligatoria

# Objetivo



- Crear nuestro propio sistema de ficheros sobre un disco virtual representado por un fichero del SF nativo de Linux
- Montar nuestro sistema de ficheros con FUSE para poder interaccionar con él con las herramientas habituales (ls, cat, nautilus, ...)
- Afianzar el conocimiento sobre el interfaz POSIX para ficheros
- Afianzar el conocimiento sobre el funcionamiento de los sistemas de ficheros

# Agenda



1 Objetivo

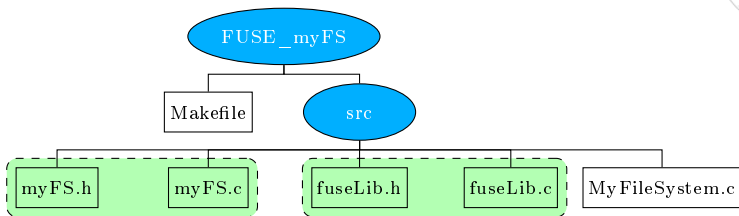
2 Proyecto FUSE\_myFS

3 myFS

4 FUSE y fuseLib

5 Parte Obligatoria

# Ficheros del proyecto



# Agenda



- 1 Objetivo
- 2 Proyecto FUSE\_myFS
- 3 myFS**
- 4 FUSE y fuseLib
- 5 Parte Obligatoria

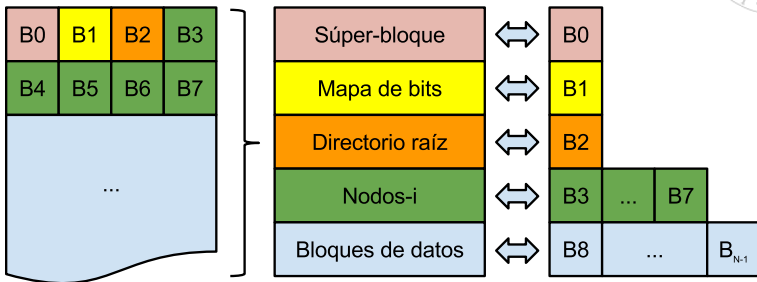
# Sistema de Ficheros Simple



Sistema de ficheros de bloques indexados tipo UNIX:

- Sólo un directorio, de tamaño 1 bloque
  - Limitamos el tamaño del nombre de los ficheros
- Estructura del nodo-i:
  - sólo enlaces directos, todo el índice en el nodo-i
- 1 bloque para el superbloque
- 1 bloque para el Mapa de bits
- Tabla de nodos-i: 5 bloques
- Resto para bloques de datos
  - $(\text{Tamaño Disco Virtual} / \text{Tamaño de bloque})$
  - Debe haber 1 como mínimo
- Limitamos el tamaño de los ficheros (en bloques)

# Estructura del disco Virtual



Archivo  $\Leftrightarrow$  SF  $\Leftrightarrow$  Conjunto de bloques

Correspondencia estructura SF  
 $\Leftrightarrow$  bloques del archivo



# Macros



```
#define BIT unsigned
#define BLOCK_SIZE_BYTES 4096
#define NUM_BITS (BLOCK_SIZE_BYTES/sizeof(BIT))
#define MAX_BLOCKS_WITH_NODES 5
#define MAX_BLOCKS_PER_FILE 100
#define MAX_FILES_PER_DIRECTORY 100
#define MAX_LEN_FILE_NAME 15
#define DISK_LBA int
#define BOOLEAN int

#define SUPERBLOCK_IDX 0
#define BITMAP_IDX 1
#define DIRECTORY_IDX 2
#define NODES_IDX 3
```

# Sistema de Ficheros



Se utiliza la siguiente estructura para representar en memoria el SF:

```
typedef struct MyFileSystemStructure {  
    int fdVirtualDisk;    // File descriptor where the whole filesystem is stored  
    SuperBlockStruct superBlock; // Super block  
    BIT bitMap[NUM_BITS]; // Bit map  
    DirectoryStruct directory; // Root directory  
    NodeStruct* nodes[MAX_NODES]; // Array of inode pointers  
    int numFreeNodes; // # of available inodes  
} MyFileSystem;
```

# Superbloque



```
typedef struct SuperBlockStructure {  
    time_t creationTime;           // Creation time  
    int diskSizeInBlocks;          // # blocks in disk  
    int numOffFreeBlocks;          // # of available blocks  
    int blockSize;                 // Block size  
    int maxLenFileName;            // Max. length of a file name  
    int maxBlocksPerFile;          // Max. number of blocks per file  
} SuperBlockStruct;
```

# Directorio



```
typedef struct FileStructure {  
    int    nodeIdX;                // Associated i-node  
    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]; // Files  
} DirectoryStruct;
```

Directory:

numFiles		
nodeIdx	fileName	freeFile
nodeIdx	fileName	freeFile
nodeIdx	fileName	freeFile
nodeIdx	fileName	freeFile
nodeIdx	fileName	freeFile

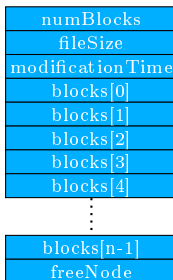
⋮

# Nodo-i

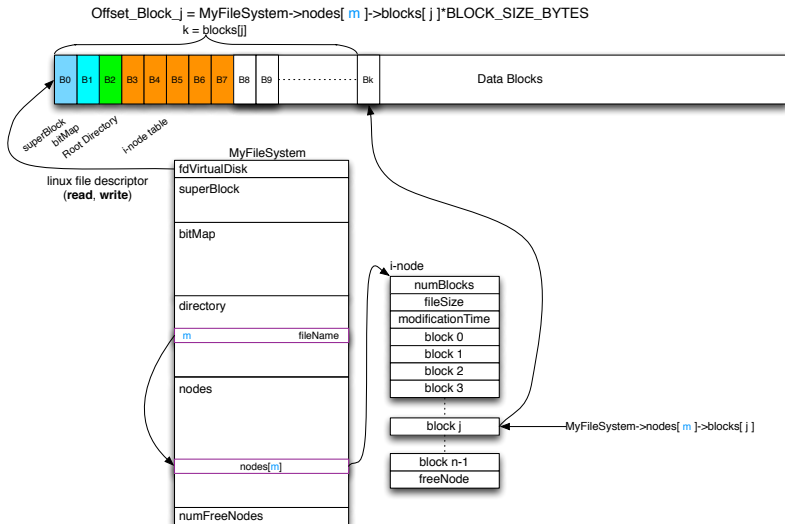


```
typedef struct NodeStructure {  
    int numBlocks;           // Num blocks  
    int fileSize;           // File size  
    time_t modificationTime; // Modification time  
    DISK_LBA blocks[MAX_BLOCKS_PER_FILE]; // Blocks  
    BOOLEAN freeNode;       // If the node is available  
} NodeStruct;
```

Nodo-i:



# Acceso a MiSistemaDeFicheros



# Funciones Manejo del SF (I)



## ■ Principales:

- `int myMkfs(MyFileSystem *myFileSystem, int diskSize, char *backupFileName)`
- `void myFree(MyFileSystem *myFileSystem)`

## ■ Escritura sobre disco virtual:

- `int updateSuperBlock(MyFileSystem *myFileSystem)`
- `int updateBitmap(MyFileSystem *myFileSystem)`
- `int updateDirectory(MyFileSystem *myFileSystem)`
- `int updateNode(MyFileSystem *myFileSystem, int nodeNum, NodeStruct *node)`

## ■ Lectura del disco virtual:

- `int readNode(MyFileSystem *myFileSystem, int nodeNum, NodeStruct* node)`

# Funciones Manejo del SF (II)



## ■ Auxiliares:

- `void copyNode(NodeStruct *dest, NodeStruct *src)`
- `int reserveBlocksForNodes(MyFileSystem* myFileSystem, DISK_LBA blockIdxs[], int numBlocks)`
- `int findNodeByPos(int nodeNum)`
- `int findFreeNode(MyFileSystem *myFileSystem)`
- `int findFileByName(MyFileSystem *myFileSystem, char *fileName)`
- `int findFreeFile(MyFileSystem *myFileSystem)`
- `void initializeSuperBlock(MyFileSystem *myFileSystem, int diskSize)`
- `int initializeNodes(MyFileSystem *myFileSystem)`
- `int myQuota(MyFileSystem *myFileSystem)`

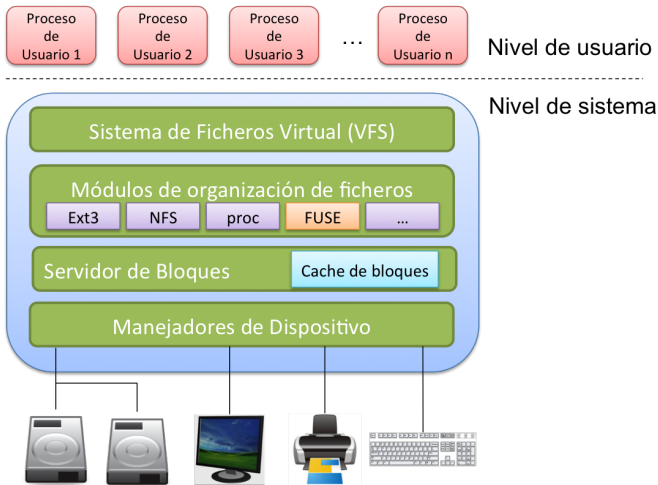


# Agenda

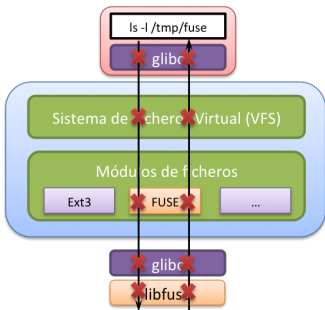


- 1 Objetivo
- 2 Proyecto FUSE\_myFS
- 3 myFS
- 4 FUSE y fuseLib**
- 5 Parte Obligatoria

# Estructura del servidor de ficheros



# FUSE: Filesystem in Userspace



- Módulo de kernel: manejador SF Fuse
- Montaje:
  - 1 solicitud proceso a módulo (/proc)
  - 2 registro SF Fuse en punto de montaje
  - 3 creación socket entre módulo y proceso
  - 4 Accesos al SF redirigidas al proceso por el socket
- Acciones realizadas por el proceso de usuario

# FUSE: ¿Cómo se usa?



- 1 Incluirémos en nuestro programa el fichero **fuse.h**
  - Declaración de tipo **struct** fuse\_operations
  - Declaración adelantada de la función fuse\_main
- 2 Declaremos una variable **struct** fuse\_operations
  - Contiene punteros a funciones que serán llamados por cada operación
- 3 La función main terminará con la llamada a fuse\_main
  - El proceso se queda atendiendo al socket
- 4 Enlazaremos nuestro programa con **libfuse**

# FUSE: fuse\_main



- Se encarga de montar el SF con libfuse
- Abre un socket para comunicarse con el módulo FUSE
  - argc: número de argumentos para el montaje
  - argv: argumentos de montaje
    - f: trabajar en primer plano
    - d: habilitar salida de depuración de FUSE (implica -f)
    - s: deshabilita multi-hilo (facilita depuración)
    - directorio: punto de montaje de FUSE
  - op: estructura con punteros a las funciones que implementan las operaciones de nuestro SF

# Ejemplo



## Creación SF con FUSE

```
> ./fs-fuse -t 2097152 -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)
504 blocks for data (2064384 B)
Formatting completed!
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
```

# FUSE: estructura fuse\_operations



```
struct fuse_operations myFS_operations = {  
    .getattr    = my_getattr,    // Obtain attributes from a file  
    .readdir    = my_readdir,    // Read directory entries  
    .truncate   = my_truncate,   // Modify the size of a file  
    .open       = my_open,       // Open a file  
    .write      = my_write,      // Write data into a file already opened  
    .release    = my_release,    // Close an open file  
    .mknod      = my_mknod,      // Create a new file  
};
```

## Normas generales:

- Las llamadas a write/read deben devolver
  - un número positivo indicando los bytes leídos
  - 0 en caso de EOF
  - número negativo en caso de error
- El resto de las funciones deben devolver
  - un número negativo en caso de error
  - 0 en otro caso

# FUSE: Operaciones



- `int (*open) (const char *, struct fuse_file_info *);`
- `int (*read) (const char *, char *, size_t, off_t, struct fuse_file_info *);`
- `int (*readdir) (const char *, void *, fuse_fill_dir_t, off_t, struct fuse_file_info *);`
- `int (*mknod) (const char *, mode_t, dev_t);`
- `int (*unlink) (const char *);`
- `int (*rename) (const char *, const char *);`
- `int (*truncate) (const char *, off_t);`
- `int (*write) (const char *, const char *, size_t, off_t, struct fuse_file_info *);`



# Agenda



- 1 Objetivo
- 2 Proyecto FUSE\_myFS
- 3 myFS
- 4 FUSE y fuseLib
- 5 Parte Obligatoria**

# ¿Qué debe hacer el alumno?



- 1 Implementar las operaciones:
  - read
  - unlink
- 2 Registrar estas operaciones en el campo correspondiente de `fuse_operations`.
- 3 Desarrollar un script de test
  - Descrito en el guión de la práctica

# Operación read: interfaz



```
int (*read) (const char *path, char *buf, size_t size, off_t offset, struct
    fuse_file_info *fi);
```

- path: nombre/ruta del fichero
- buf: buffer donde hay que copiar los bytes leídos del fichero
- size: cantidad de bytes a leer
- offset: offset desde el comienzo del fichero para comenzar la lectura
- fi: estructura de FUSE asociada al fichero
- Devuelve: un número negativo en caso de error, 0 en otro caso

# Operación read: pseudocódigo



```
char buffer[BLOCK_SIZE_BYTES];  
int bytes2Read, totalRead = 0;
```

Comprobar que offset no sale fuera del fichero

bytes2Read = numero de bytes a leer (puede ser menor que size)

```
while (totalRead < bytes2Read) {  
    int i, currentBlock, offBlock;  
  
    currentBlock = bloque físico en la posición offset  
    offBlock      = desplazamiento en el bloque físico correspondiente  
  
    buffer ← bloque currentblock del disco virtual  
  
    copiar los bytes leídos al buffer de salida buf  
  
    actualizar offset y totalRead  
}  
  
return totalRead;
```

# Operación unlink: pseudocódigo



```
static int my_unlink(const char *path) {  
    int idxNode;
```

Buscar path en el directorio del SF

idxNode = nodo-i del fichero

Truncar el fichero utilizando **resizeNode**

Marcar la entrada de directorio como libre

Decrementar el contador de ficheros del directorio

Marcar el nodo-i como libre

Incrementar el contador de nodos-i libres

Actualizar el directorio en el disco virtual

Actualizar el nodo-i en el disco virtual

Liberar la memoria del nodo-i y actualizar la tabla

```
    return 0;
```

```
}
```