Sistemas Operativos
Práctica 2

Curso

2016-2017

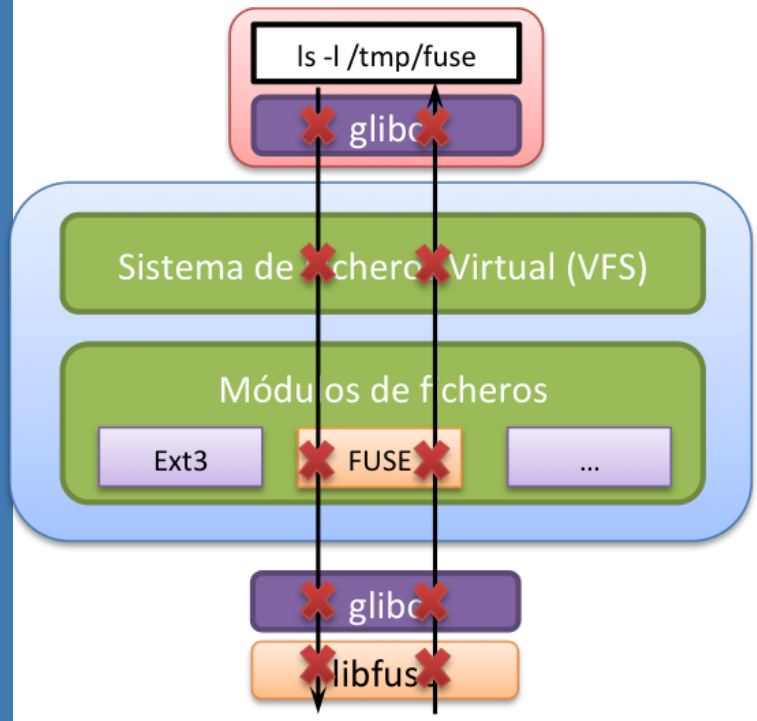
Sistema de Ficheros

Joaquín Recas

Objetivos

- 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 interacicionar con él con las herramientas habituales (ls, cat, nautilus, . . .)

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. Creamos programa principal (achivo .c)
   - Hay que incluir `fuse.h` y enlazar con `libfuse`
2. Declarar estructura llamada `fuse_operations`
   - Contiene punteros a funciones que serán llamados por cada operación (*callbacks*)
3. Se invoca `fuse_main`
   - El proceso se queda atendiendo al `socket`

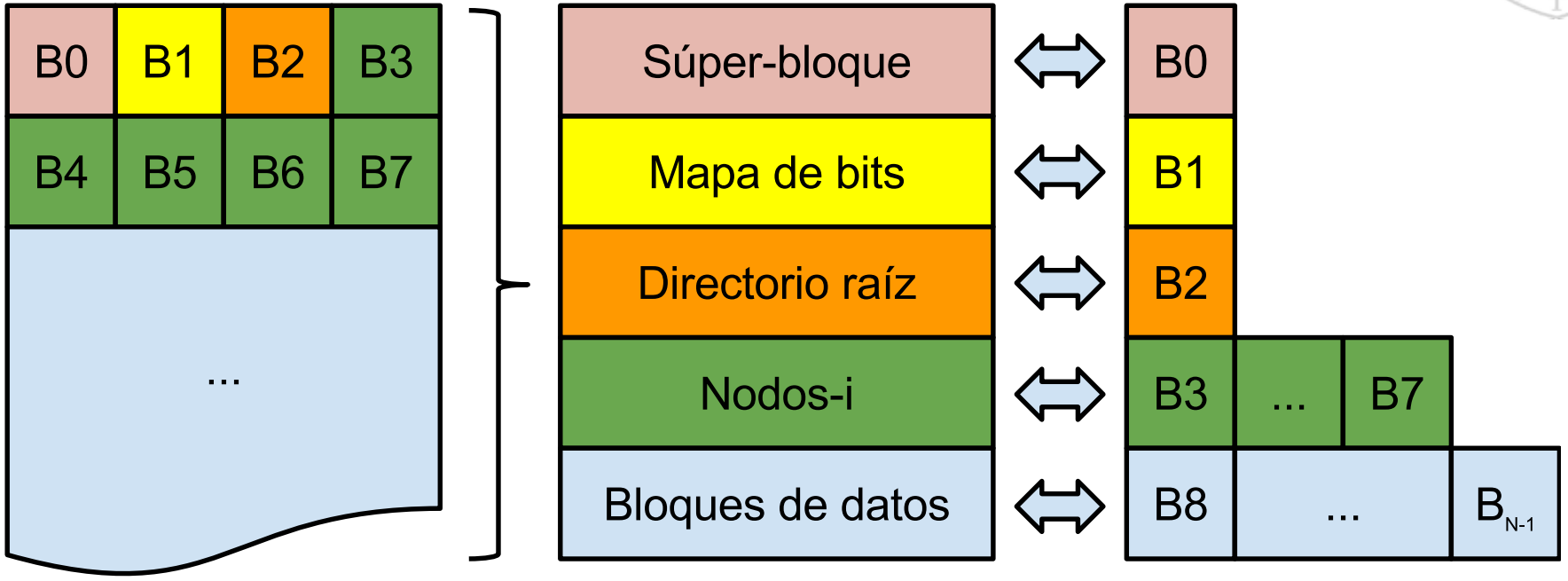
FUSE: fuse_operations

- `int (*getattr)(const char *, struct stat *)`
- `int (*readdir)(const char *, void *, fuse_fill_dir_t, off_t, struct fuse_file_info *)`
- `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 (*mknod)(const char *, mode_t, dev_t)`
- `int (*unlink) (const char *)`

Simplificaciones del SF

- 1 bloque para el superbloque
- 1 bloque para el Mapa de bits
- Sólo un directorio (de tamaño 1 bloque)
- 5 bloques para ls tabla de nodos-i
  - Sólo enlaces directos en el nodo-i
  - Límite en el tamaño de los ficheros (en bloques)
- Resto para bloques de datos:
  - Tamaño Disco Virtual / Tamaño de bloque
  - Debe haber 1 como mínimo

Estructura del disco Virtual



Archivo ⇔ SF ⇔ Conjunto de bloques

Correspondencia estructura SF ⇔ bloques del archivo

myFS.h (I)

Definiciones

```
#define false 0
#define true 1

#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
```

myFS.h (II)

Superbloque

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

myFS.h (III)

Directorio

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

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

myFS.h (IV)

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 avail.
} NodeStruct;
```

myFS.h (V)

Sistema de Ficheros

```
#define NODES_PER_BLOCK (BLOCK_SIZE_BYTES/sizeof(NodeStruct))
#define MAX_NODES (NODES_PER_BLOCK * MAX_BLOCKS_WITH_NODES)

typedef struct MyFileSystemStructure {
  // File descriptor where the whole filesystem is stored
  int fdVirtualDisk;
  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;
```

Funciones Manejo del SF (I)

- 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)`
  - `int readBlock(MyFileSystem *myFileSystem, DISK_LBA blockNum, void *buff)`
  - `int writeBlock(MyFileSystem *myFileSystem, DISK_LBA blockNum, void *buff)`

Funciones Manejo del SF (II)

- Funciones auxiliares:
  - `int findFileByName(MyFileSystem *myFileSystem, char *fileName)`
  - `int resizeNode(uint64_t idxNode, size_t newSize)`
  - `int findFreeFile(MyFileSystem *myFileSystem)`
  - `int findFreeNode(MyFileSystem *myFileSystem)`
  - `int reserveBlocksForNodes(MyFileSystem* myFileSystem, DISK_LBA blckIdxs[], int numBlocks)`

Funciones Manejo del SF (III)

- Miscelánea:
  - `int myMkfs(MyFileSystem *myFileSystem, int diskSize, char *backupFileName)`
  - `void initializeSuperBlock(MyFileSystem *myFileSystem, int diskSize)`
  - `int initializeNodes(MyFileSystem *myFileSystem)`
  - `void copyNode(NodeStruct *dest, NodeStruct *src))`
  - `int findNodeByPos(int nodeNum)`
  - `void myFree(MyFileSystem *myFileSystem)`
  - `int myMount(MyFileSystem *myFileSystem, char *backupFileName)`
  - `int myQuota(MiSistemaDeFicheros* miSistemaDeFicheros)`

Ejecución de la Práctica

- Argumentos:
  - `t`: tamaño en bytes del SF
  - `a`: fichero que contendrá nuestro SF
  - `f`: argumentos a `FUSE`
- Ejemplo:

```
./MiSistemaDeFicheros -t 2097152 -a disco-virtual -f '-d -s punto-montaje'
```

  - Monta nuestro sistema de ficheros en un directorio que deberá de estar vacío
  - Podremos interactuar con nuestro sistema de ficheros con los comandos habituales (ls, cat, nautilus, ...)

¿Qué debe hacer el alumno?

- Implementar las siguientes operaciones:
  - `unlink`:
    - Prototipo en manual de FUSE
    - Manual de `errno` para asignar el valor de retorno.
  - `read`:
    - Devuelve el mínimo entre el número de bytes del fichero y el número de bytes solicitados (0 si no hay más datos).
    - Valor negativo en caso de error.
- Registrar estas operaciones en el campo correspondiente de `fuse_operations`.
- Desarrollar un `script` de test (ver guión)