

# Sistemas Operativos

Curso  
2014-2015

## Práctica 1

*Introducción al entorno de desarrollo*

J.C. Sáez

# Introducción

## Objetivos

- Familiarizarse con el entorno de desarrollo de aplicaciones C en LINUX
- Familiarizarse con el manejo básico del shell y aprender a desarrollar *scripts* sencillos

## Requisitos

- Leer los siguientes documentos:
  - Introducción al entorno de desarrollo
  - Revisión: Programación en C
  - Introducción al shell Bash
  - Manual descriptivo “Entorno de desarrollo C para GNU/Linux”

# Archivo mtar

**Archivo mtar:** fichero binario que alberga múltiples ficheros en su interior

|                        |
|------------------------|
| Número de ficheros (N) |
| ruta fichero 1         |
| tamaño fichero 1       |
| ruta fichero 2         |
| tamaño fichero 2       |
| ...                    |
| ruta fichero N         |
| tamaño fichero N       |
| datos fichero 1        |
| datos fichero 2        |
| ...                    |
| datos fichero N        |



# Programa mitar

## Modo de uso

```
mitar -c|x -f archivo_mtar [fich1 fich2 ...]
```

- -c : Crear archivo mtar

- Ejemplo: `./mitar -c -f ejemplo.mtar a.txt b.txt`

- -x : Extraer archivo mtar

- Ejemplo: `./mitar -x -f ejemplo.mtar`

# Implementación (I)

## Proyecto proporcionado

- El proyecto consta de los siguientes ficheros:
  - `makefile`
  - `mitar.c` : función `main()` del programa
    - El procesamiento de opciones de la línea de comandos está ya implementado
  - `mitar.h` : declaraciones de tipos de datos y funciones
  - `rut_mitar.c` : funciones de creación y extracción de ficheros mtar
    - Único fichero a modificar

# Implementación (II)

## mitar.h

```
#ifndef _MITAR_H
#define _MITAR_H

#include <limits.h>

typedef enum{
    NONE,
    ERROR,
    CREATE,
    EXTRACT
} flags;

typedef struct {
    char* name;
    unsigned int size;
} stHeaderEntry;

int createTar(int nFiles, char *fileNames[], char tarName[]);
int extractTar(char tarName[]);

#endif /* _MITAR_H */
```

# Implementación (III)

## Funciones a implementar (rut\_mitar.c)

- `int createTar(int nFiles, char *fileNames[], char* tarName);`
  - Crea un fichero mtar con nombre 'tarName' incluyendo en él los ficheros cuya rutas están especificadas en el array `fileNames`
- `int extractTar(char* tarName);`
  - Extrae el fichero mtar cuya ruta se pasa como parámetro
- `int copynFile(FILE *origen, FILE *destino, int nBytes);`
  - Transfiere `nBytes` del fichero origen al fichero destino
    - La transferencia se ha de realizar byte a byte usando `getc()` y `putc()`
    - La copia de datos finalizará al transferir `nBytes` o cuando se llegue al fin del fichero origen
    - Para forzar copia hasta fin de fichero → pasar `INT_MAX` como tercer parámetro de la función (macro definida en `<limits.h>`)
  - `copynFile()` devuelve el número de bytes que se han transferido realmente

# Implementación (IV)

## Funciones a implementar (rut\_mitar.c)

- `int readHeader(FILE *tarFile, stHeaderEntry **header, int *nFiles);`
  - Lee la cabecera del fichero mtar `tarFile` y copia la metainformación en el array `header`
    - La función ha de reservar memoria para el array `header` (de ahí el doble puntero → puntero por referencia)
  - Devuelve en `nFiles` (entero por referencia) el número de ficheros contenidos en el mtar
- `int loadstr( FILE *file, char** buf );`
  - Lee una cadena de caracteres del fichero cuyo descriptor se pasa como parámetro
  - La función reserva memoria para la cadena leída. La dirección de memoria donde comienza la cadena se devuelve a la función invocadora mediante el parámetro `buf`



# Implementación (V)

## Uso del doble puntero en readHeader()

```
int readHeader(FILE *tarFile, stHeaderEntry **header, int *nFiles)
{
    stHeaderEntry* array=NULL;
    int nr_files=0;

    ... Leemos el número de ficheros (N) del tarFile y lo volcamos en nr_files ...

    /* Reservamos memoria para el array */
    array=malloc(sizeof(stHeaderEntry)*nr_files);

    ... Leemos la metainformación del tarFile y la volcamos en el array ...

    /* Devolvemos los valores leídos a la función invocadora */
    (*nFiles)=nr_files;
    (*header)=array;

    return (EXIT_SUCCESS);
}
```

# Creación de un fichero mtar (I)

- La creación de un fichero mtar **exige realizar escrituras en el fichero en desorden**
  - No sabemos de antemano cuál es el tamaño en bytes de cada uno de los ficheros que hay que introducir en el mtar
  - Solo sabremos el tamaño de cada archivo una vez lo hayamos leído por completo y transferido su contenido al fichero mtar vía `copynFile()`

# Ejemplo: Creación de un fichero mtar

```
$ ./mitar -c -f test.mtar a.txt b.txt c.txt
```



# Ejemplo: Creación de un fichero mtar

```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

**Archivo test.mtar**  
(en disco)



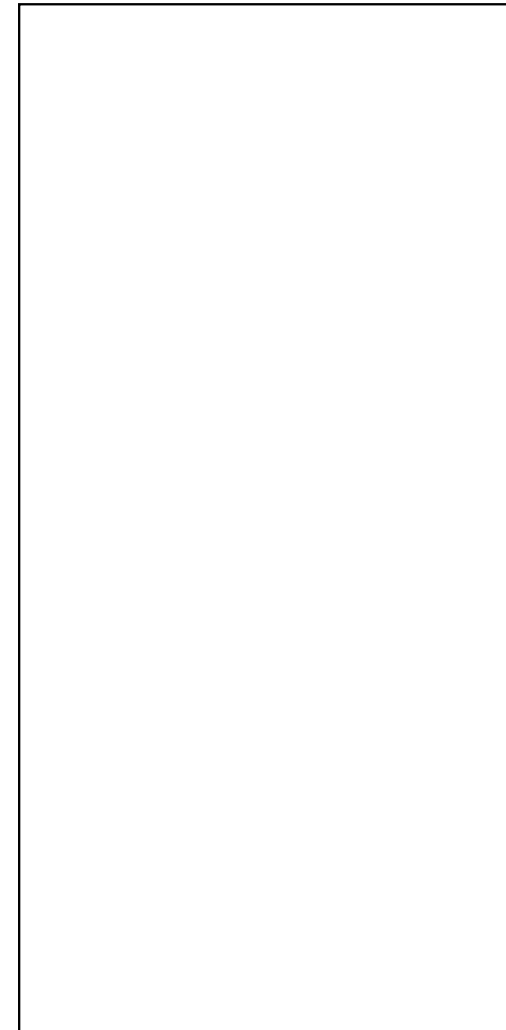
# Ejemplo: Creación de un fichero mtar

```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

**Archivo test.mtar**  
(en disco)

**Array de stHeaderEntry**  
(en memoria)

|     |    |
|-----|----|
| [0] | ?? |
|     | ?? |
| [1] | ?? |
|     | ?? |
| [2] | ?? |
|     | ?? |

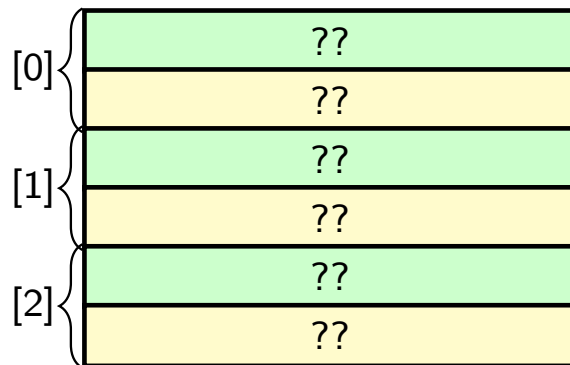


# Ejemplo: Creación de un fichero mtar

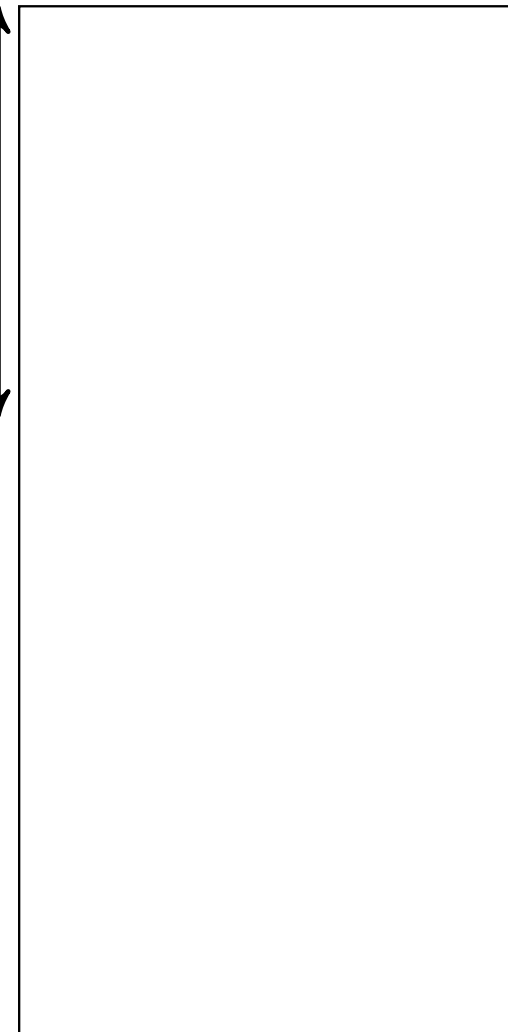
```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

**Archivo test.mtar**  
(en disco)

**Array de stHeaderEntry**  
(en memoria)



$$\text{sizeof(int)} + \text{nFiles} * \text{sizeof(unsigned int)} + \sum_{i=0}^{\text{nFiles}-1} (\text{strlen}(\text{fileNames}[i]) + 1)$$



# Ejemplo: Creación de un fichero mtar

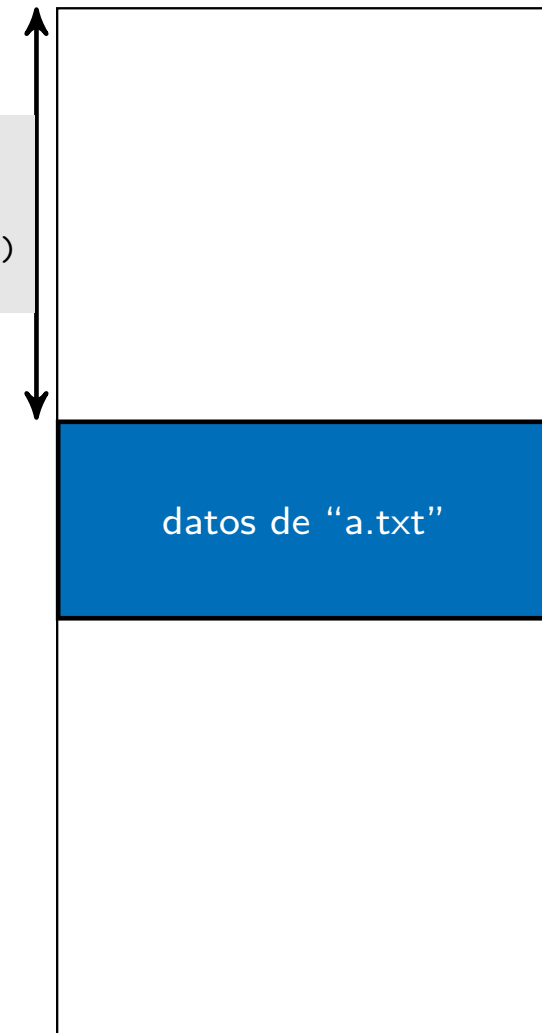
```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

**Archivo test.mtar**  
(en disco)

**Array de stHeaderEntry**  
(en memoria)

|     |         |
|-----|---------|
| [0] | "a.txt" |
|     | 7       |
| [1] | ??      |
|     | ??      |
| [2] | ??      |
|     | ??      |

$$\text{sizeof(int)} + nFiles * \text{sizeof(unsigned int)} + \sum_{i=0}^{nFiles-1} (\text{strlen}(\text{fileNames}[i]) + 1)$$



# Ejemplo: Creación de un fichero mtar

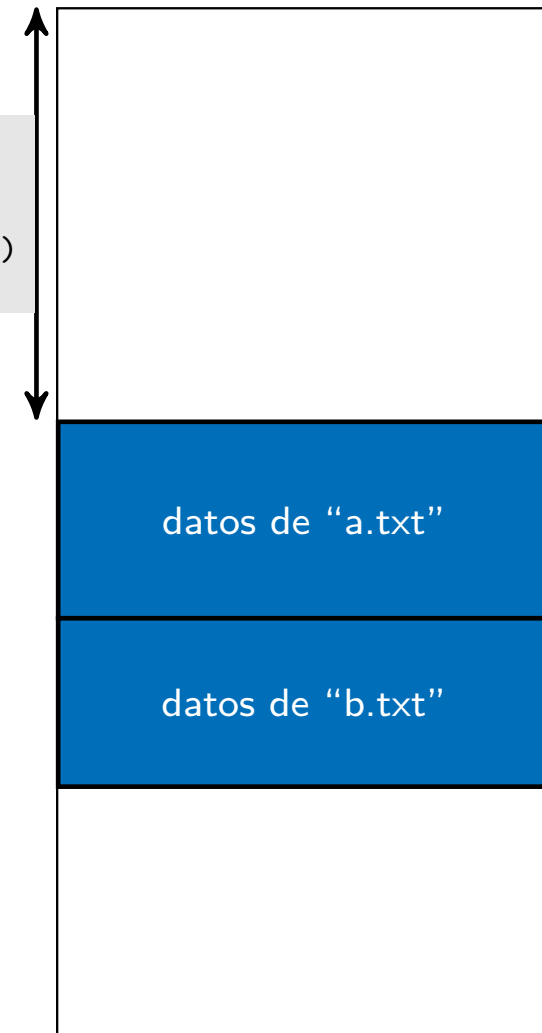
```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

**Archivo test.mtar**  
(en disco)

**Array de stHeaderEntry**  
(en memoria)

|     |         |
|-----|---------|
| [0] | "a.txt" |
|     | 7       |
| [1] | "b.txt" |
|     | 6       |
| [2] | ??      |
|     | ??      |

$$\text{sizeof(int)} + nFiles * \text{sizeof(unsigned int)} + \sum_{i=0}^{nFiles-1} (\text{strlen}(\text{fileNames}[i]) + 1)$$





# Ejemplo: Creación de un fichero mtar

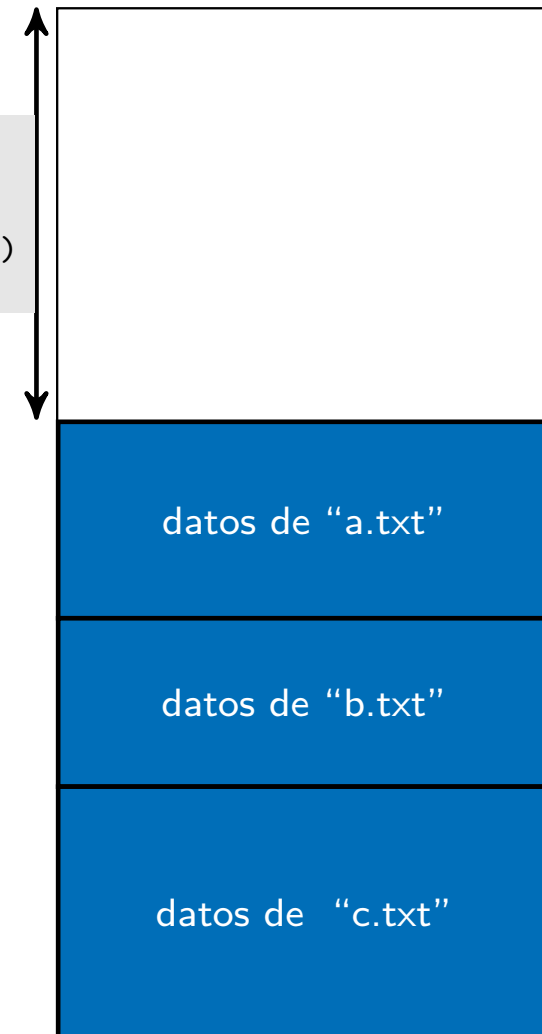
```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

**Archivo test.mtar**  
(en disco)

**Array de stHeaderEntry**  
(en memoria)

|     |         |
|-----|---------|
| [0] | "a.txt" |
|     | 7       |
| [1] | "b.txt" |
|     | 6       |
| [2] | "c.txt" |
|     | 9       |

$$\text{sizeof(int)} + nFiles * \text{sizeof(unsigned int)} + \sum_{i=0}^{nFiles-1} (\text{strlen}(\text{fileNames}[i]) + 1)$$



# Ejemplo: Creación de un fichero mtar

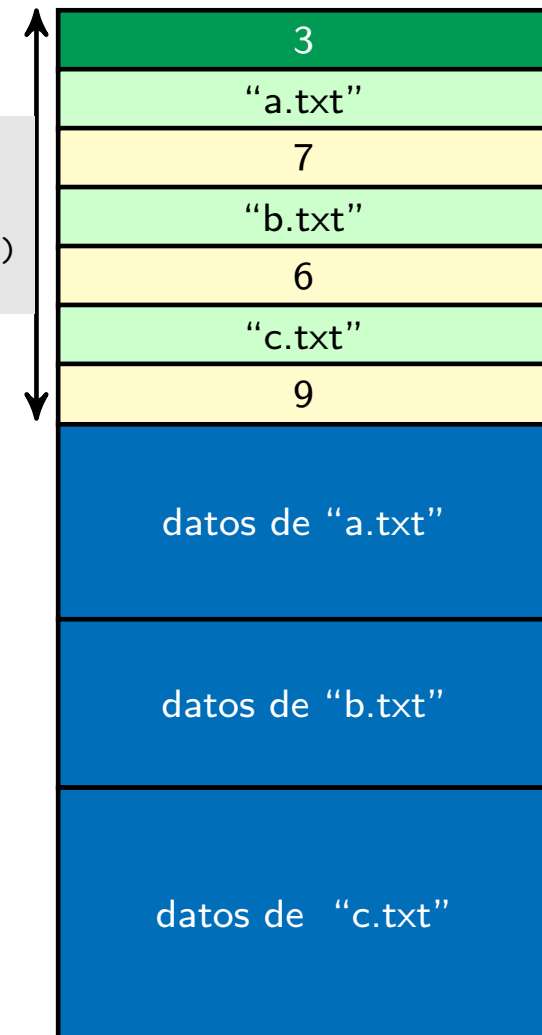
```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

**Archivo test.mtar**  
(en disco)

**Array de stHeaderEntry**  
(en memoria)

|     |         |
|-----|---------|
| [0] | "a.txt" |
|     | 7       |
| [1] | "b.txt" |
|     | 6       |
| [2] | "c.txt" |
|     | 9       |

$$\text{sizeof(int)} + n\text{Files} * \text{sizeof(unsigned int)} + \sum_{i=0}^{n\text{Files}-1} (\text{strlen}(\text{fileNames}[i]) + 1)$$



# Creación de un fichero mtar (II)

## Pasos a llevar a cabo en createTar()

- Abrimos el fichero mtar para escritura (fichero destino)
- Reservamos memoria (con `malloc()`) para un array de `stHeaderEntry`
  - El array tendrá tantas posiciones como ficheros haya que introducir en el mtar
- Inicializar campo `name` de cada estructura `stHeaderEntry`
  - Exige reservar memoria para alojar la cadena asociada a cada nombre de fichero (No olvidar reservar espacio para el `'\0'`)
- Nos posicionamos en el byte del fichero donde comienza la región de datos:
 
$$\text{sizeof(int)} + n\text{Files} * \text{sizeof(unsigned int)} + \sum_{i=0}^{n\text{Files}-1} (\text{strlen}(\text{fileNames}[i]) + 1))$$
  - De este modo dejamos hueco para el número de ficheros y los metadatos de cada uno (ruta,tamaño)

# Creación de un fichero mtar (II)

## Pasos a llevar a cabo en createTar() (cont.)

- Por cada fichero (inputFile) que haya que copiar en el mtar:
  - 1 Abrimos inputFile
  - 2 `copynFile(inputFile, tarFile, INT_MAX)`
  - 3 Cerramos inputFile
  - 4 Rellenamos el elemento correspondiente del array de estructuras con el tamaño del fichero que acabamos de volcar a disco
- Nos posicionamos para escribir en el byte 0 del fichero tar para:
  - 1 escribir número de ficheros en el fichero (4 bytes)
  - 2 Para cada estructura `stHeaderEntry`:
    - escribir la ruta del fichero (con '`\0`' al final)
    - escribir el número de bytes que ocupa el fichero
- Liberamos memoria y cerramos el fichero mtar

# Ejemplo de ejecución

terminal

```
usuario@debian6:~/Temp/Mitar$ ls
a.txt  b.txt  c.txt  makefile  mitar.c  mitar.h  rut_mitar.c
usuario@debian6:~/Temp/Mitar$ du -b *.txt
7    a.txt
6    b.txt
9    c.txt
usuario@debian6:~/Temp/Mitar$ make
gcc -g -Wall -c mitar.c -o mitar.o
gcc -g -Wall -c rut_mitar.c -o rut_mitar.o
gcc -g -Wall -o mitar mitar.o rut_mitar.o
usuario@debian6:~/Temp/Mitar$ ./mitar -c -f test.mtar a.txt b.txt c.txt
Fichero mitar creado con exito
usuario@debian6:~/Temp/Mitar$ ls
a.txt  c.txt      mitar      mitar.h  rut_mitar.c  test.mtar
b.txt  makefile  mitar.c    mitar.o  rut_mitar.o
```

## Ejemplo de ejecución (cont.)

terminal

```
usuario@debian6:~/Temp/Mitar$ mkdir tmp
usuario@debian6:~/Temp/Mitar$ cd tmp/
usuario@debian6:~/Temp/Mitar/tmp$ ../mitar -x -f ../test.mtar
[0]: Creando fichero a.txt, tamaño 7 Bytes...0k
[1]: Creando fichero b.txt, tamaño 6 Bytes...0k
[2]: Creando fichero c.txt, tamaño 9 Bytes...0k
usuario@debian6:~/Temp/Mitar/tmp$ ls
a.txt  b.txt  c.txt
usuario@debian6:~/Temp/Mitar/tmp$ diff a.txt ../a.txt
usuario@debian6:~/Temp/Mitar/tmp$ diff b.txt ../b.txt
usuario@debian6:~/Temp/Mitar/tmp$ diff c.txt ../c.txt
usuario@debian6:~/Temp/Mitar/tmp$
```

# Entrega de la práctica

- Hasta el **20 de octubre 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)

