

Práctica 1

Curso
2016-2017

Introducción al entorno de desarrollo

Joaquín Recas

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

Objetivos

- 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: ./mytar -c -f ejemplo.mtar a.txt b.txt

- -x : Extraer archivo mtar

- Ejemplo: ./mytar -x -f ejemplo.mtar

Implementación (I)

Proyecto proporcionado

- El proyecto consta de los siguientes ficheros:
 - makefile
 - mytar.c : función `main()` del programa
 - mytar.h : declaraciones de tipos de datos y funciones
 - mytar_routines.c : funciones de creación y extracción de ficheros mtar
- Sólo se modificará el fichero `mytar_routines.c`

Implementación (II)



mytar.h

```
#ifndef _MYTAR_H
#define _MYTAR_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 /* _MYTAR_H */
```

Implementación (III)

Funciones a implementar (mytar_routines.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

Implementación (IV)



Funciones a implementar (mytar_routines.c)

- `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 cuando se transfieran `nBytes` o se llegue al fin del fichero origen
 - Para forzar copia hasta fin de fichero, haremos que `nBytes` sea `INT_MAX` (macro definida en `<limits.h>`)
 - `copynFile()` devuelve el número de bytes que se han transferido realmente

Implementación (V)

Funciones a implementar (`mytar_routines.c`)

- `stHeaderEntry* readHeader(FILE *tarFile, int *nFiles);`
 - Lee la cabecera del fichero `mtar tarFile` y retorna el array de pares (nombre,tamaño)
 - La memoria para el array ha de reservarse con `malloc()` en el interior de esa función
 - Devuelve en `nFiles` (entero por referencia) el número de ficheros contenidos en el `mtar`
- `char* loadstr(FILE *file);`
 - Lee una cadena de caracteres del fichero cuyo descriptor se pasa como parámetro
 - Usar esta función en la implementación de `readHeader()`
 - La función reserva memoria para la cadena leída. La dirección de memoria donde comienza la cadena se devuelve como valor de retorno.

Implementación (IV)



Uso del doble puntero en readHeader()

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

    /* ... Read the number of files (N) from tarfile and
       store it in nr_files ... */

    /* Allocate memory for the array */
    array=malloc(sizeof(stHeaderEntry)*nr_files);

    /*... Read the (pathname,size) pairs from tarFile and
       store them in the array ...*/

    /* Store the number of files in the output parameter */
    (*nFiles)=nr_files;

    return array;
}
```

Creación de fichero mtar (I)

- La creación de un fichero mtar la realizaremos mediante **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 al fichero mtar vía `copynFile()`

Creación de fichero mtar (II)

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

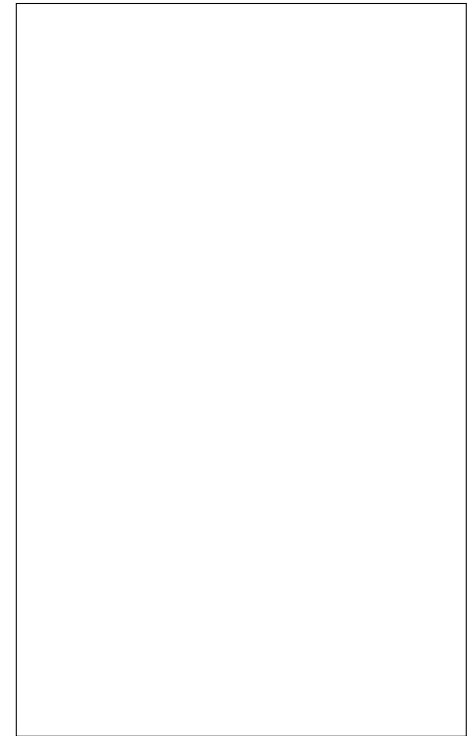


Creación de fichero mtar (II)



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

Archivo test.mtar
(en disco)



Creación de fichero mtar (II)

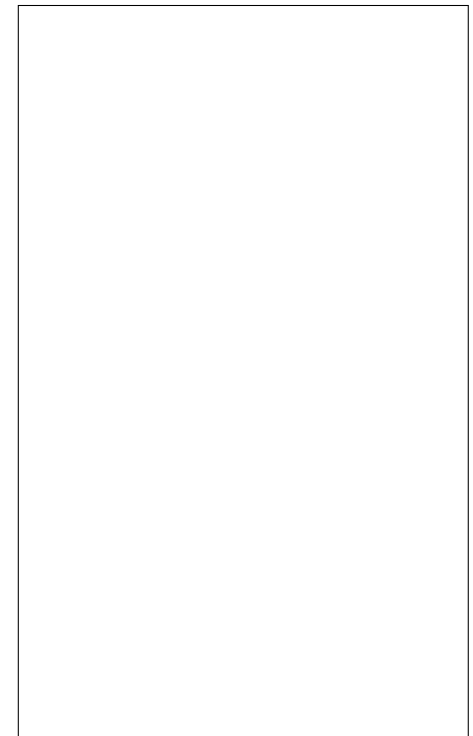


```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

Array de stHeaderEntry
(en memoria)

[0]	??
	??
[1]	??
	??
[2]	??
	??

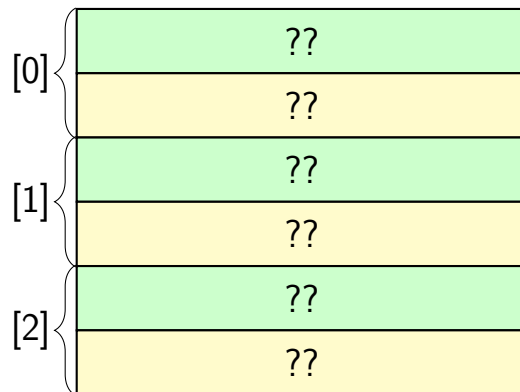
Archivo test.mtar
(en disco)



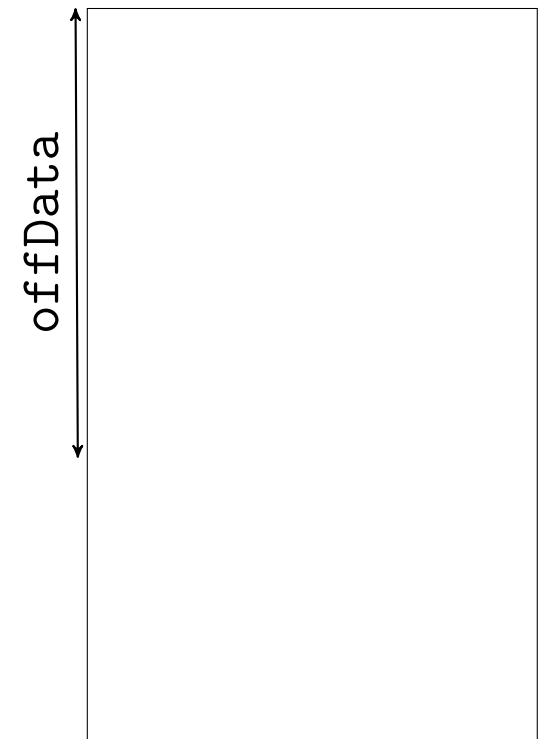
Creación de fichero mtar (II)

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

Array de stHeaderEntry
(en memoria)



Archivo test.mtar
(en disco)



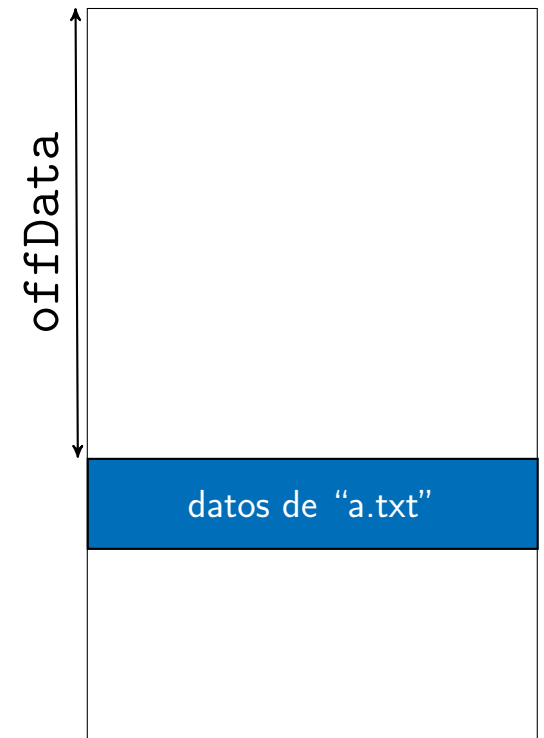
Creación de fichero mtar (II)

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

Array de stHeaderEntry
(en memoria)

[0]	Puntero a "a.txt"
	7
[1]	??
	??
[2]	??
	??

Archivo test.mtar
(en disco)



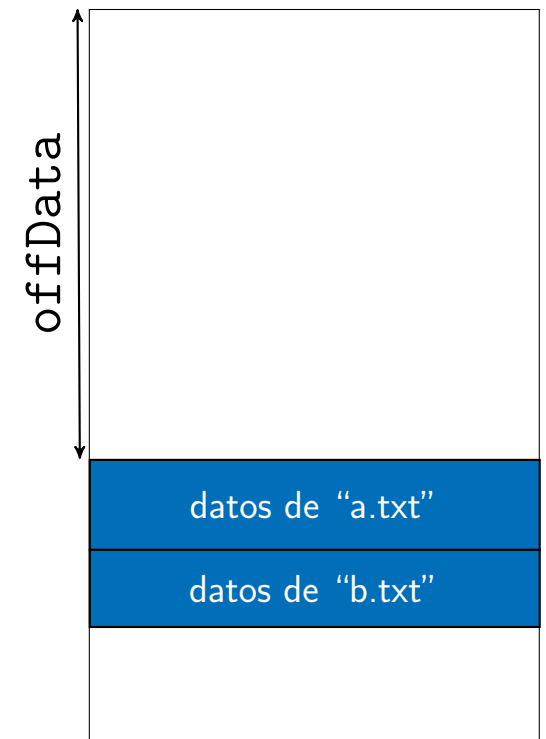
Creación de fichero mtar (II)

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

Array de stHeaderEntry
(en memoria)

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

Archivo test.mtar
(en disco)



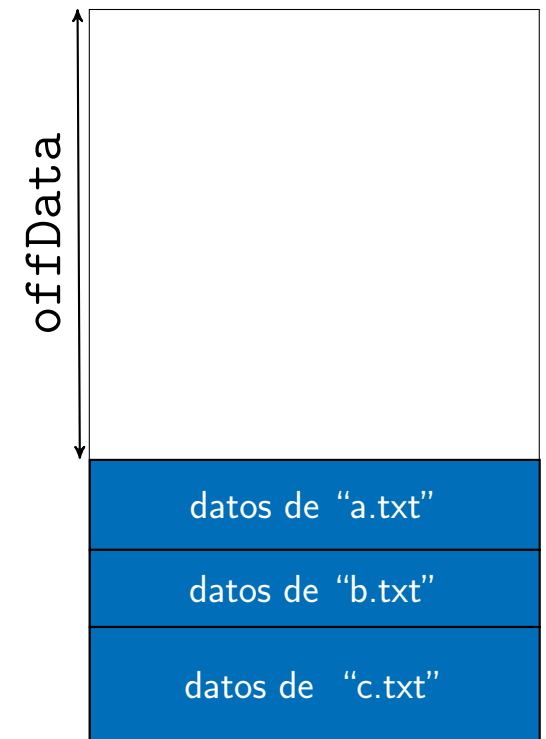
Creación de fichero mtar (II)

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

Array de stHeaderEntry
(en memoria)

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

Archivo test.mtar
(en disco)



Creación de fichero mtar (II)

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

Array de stHeaderEntry
(en memoria)

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

Archivo test.mtar
(en disco)

offData	3
	"a.txt"
	7
	"b.txt"
	6
	"c.txt"
	9
	datos de "a.txt"
	datos de "b.txt"
	datos de "c.txt"

Creación de fichero mtar (III)

Pasos a llevar a cabo en `createTar()`

- Abrimos el fichero mtar para escritura (fichero destino)
- Reservamos memoria (`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:

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

- De este modo dejamos hueco para el número de ficheros y los metadatos de cada uno (ruta,tamaño)

Creación de fichero mtar (IV)

Pasos a llevar a cabo en createTar()

- Por cada fichero (inputFile) que haya que copiar en el mtar:
 - Abrimos inputFile
 - `copynFile(inputFile, tarFile, INT_MAX)`
 - Cerramos inputFile
 - 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:
 - escribir número de ficheros en el fichero (4 bytes)
 - 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 (I)



terminal

```
osuser@debian:~/Mytar$ ls
a.txt b.txt c.txt makefile mytar.c mytar.h mytar_routines.c
osuser@debian:~/Mytar$ du -b *.txt
7      a.txt
6      b.txt
9      c.txt
osuser@debian:~/Mytar$ make
gcc -g -Wall -c mytar.c -o mytar.o
gcc -g -Wall -c mytar_routines.c -o mytar_routines.o
gcc -g -Wall -o mytar mytar.o mytar_routines.o
osuser@debian:~/Mytar$ ./mytar -c -f test.mtar a.txt b.txt c.txt
Mtar file created successfully
osuser@debian:~/Mytar$ ls
a.txt c.txt      mytar  mytar.h  mytar_routines.c  test.mtar
b.txt makefile mytar.c mytar.o  mytar_routines.o
```

Ejemplo de ejecución (II)



terminal

```
osuser@debian:~/Mytar$ mkdir tmp
osuser@debian:~/Mytar$ cd tmp/
osuser@debian:~/Mytar/tmp$ ../mytar -x -f ../test.mtar
[0]: Creating file a.txt, size 7 Bytes...0k
[1]: Creating file b.txt, size 6 Bytes...0k
[2]: Creating file c.txt, size 9 Bytes...0k
osuser@debian:~/Temp/Mytar/tmp$ ls
a.txt b.txt c.txt
osuser@debian:~/Mytar/tmp$ diff a.txt ../a.txt
osuser@debian:~/Mytar/tmp$ diff b.txt ../b.txt
osuser@debian:~/Mytar/tmp$ diff c.txt ../c.txt
osuser@debian:~/Mytar/tmp$
```

Visualizando un mtar

- Es posible usar un editor hexadecimal, como ghex2 o xxd para visualizar el contenido de un fichero mtar
 - Esto permite detectar problemas en el fichero a simple vista
- Cada línea en la salida de xxd muestra 16 bytes tanto en formato hexadecimal como en ASCII
 - Los primeros 4 bytes codifican el número de ficheros en el archivo
 - Nótese que x86 es una arquitectura *little-endian*

terminal

```
osuser@debian:~/Mytar$ xxd test.mtar
0000000: 0300 0000 612e 7478 7400 0700 0000 622e  ....a.txt....b.
0000010: 7478 7400 0600 0000 632e 7478 7400 0900  txt....c.txt...
0000020: 0000 6161 6161 6161 6162 6262 6262 6263  ..aaaaaaabbbbbbc
0000030: 6363 6363 6363 6363  cccccccc
```


Entrega de la práctica

- Hasta el **1 de Marzo a las 16:30h**
- 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 y probar la práctica (fuentes + Makefile + script de comprobación)
 - Debe incluir un fichero Leeme.txt con los nombres de los alumnos, puesto, laboratorio y posibles comentarios.
 - 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
 - Ejemplo: L03_P17_Pr1.tar.gz
- El proyecto debe compilar sin errores ejecutando make