Fundamentos de Software de Comunicaciones

Tema 2

Programación del Sistema Operativo

(1ª parte)

Contenidos

- Ficheros binarios en Linux
- Llamadas al sistema
- La tabla de descriptores de ficheros
- Información detallada de un fichero
- Manejo de directorios

Ficheros binarios en Linux

- Trabajo con ficheros en Programación I y II: C++
  - Ficheros de texto
  - Al escribir, se ven los caracteres

```
 6  int main() {
 7      ofstream salida;
 8
 9      salida.open("data.txt");
10
11      int v = 25;
12      salida << v << endl;
13
14      salida.close();
15
16      return 0;
17  }
```

- FSC
  - Ficheros binarios
  - Se vuelca en el fichero la representación en memoria de la variable

```
int v = 0x00000025; // v = 25;
write(fd, &v, sizeof(v));
```

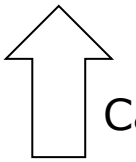
  - Caracteres no visibles
  - Representación más compacta

```
int v = 67736735; // 4 bytes en binario, 8 en formato texto
```

Ficheros binarios en Linux

- Abstracción de fichero:
  - Flujo (array) de bytes contiguos que están en el almacenamiento secundario (disco duro)

'h'	'o'	'l'	'a'	...		EOF	...	...	...



Cabezal de lectura/escritura (desplazamiento u offset)

  - Trabajo con ficheros:
    - Apertura
      - Un fichero se abre para lectura/escritura o ambos
      - El cabezal de lectura apunta a la primera posición
    - Lectura/escritura
      - A partir de la posición del cabezal, se lee o escribe
      - La posición del mismo se actualiza a la primera posición

Entrada/Salida y Ficheros

- E/S mediante **llamadas al sistema**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

  - Llamadas al sistema: open/close/creat/read/write
  - Consultar manual. Ejemplo: ***man 2 open***

- E/S mediante funciones de **librería de C/C++**
  - C:
    - #include <stdio.h>
    - Funciones: fopen/fclose/fread/fwrite/fgets/fgetc/fscanf
    - Consultar manual. Ejemplo: *man 3 fopen*
  - C++:
    - #include <fstream>
    - Clases: ifstream, ofstream

Llamadas al sistema para E/S

- Para abrir un fichero:

```
int open(const char *path, int oflag);
```

- Para crear un fichero:

```
int open(const char *path, int oflag, mode_t mode);
```

- Valor devuelto:
  - **-1 si la apertura no ha sido correcta**
  - **El descriptor de fichero en otro caso**

- **oflag** es el OR a nivel de bit (operador |) de los indicadores

  O_RDONLY O_WRONLY O_RDWR O_APPEND O_TRUNC O_CREAT

- Si se utiliza el flag **O_CREAT** se necesita un tercer argumento (*mode*) donde se especifican los permisos del fichero (ver ejemplos)

Llamadas al sistema para E/S

```
int fd = open("/home/alumno/data.dat", O_RDONLY)
if (fd < 0) {
  perror("open");
  exit(-1);
}
```

- Rutas a ficheros:
  - Absolutas
    - Empiezan por /, que es el directorio raíz
    - Menos habitual porque hace el código dependiente de una jerarquía de directorios
  - Relativas:

```
fd = open("data.dat", O_RDONLY)
```

    - Se busca el fichero en el directorio actual
    - El . (punto) es el directorio actual
    - .. (dos puntos) es el directorio padre

Llamadas al sistema para E/S

```
  int fd, fd2;                    /* descriptores de fichero  */
   ...

/*Flags para abrir un fichero que ya existe: */
if ( (fd = open("ejemplo", O_RDONLY)) < 0) {
   perror("open");
   exit(1);
}

/*Flags para crear un fichero: */
if ((fd = open("ejemplo", O_WRONLY | O_TRUNC | O_CREAT, 0600)) < 0)
{
    perror("open"););
   exit(1);
}
```

Llamadas al sistema para E/S

```
/* Descriptores de ficheros */
int fd_read;
int fd_write;
int fd_readwrite;
int fd_append;

/* Abre el fichero /etc/passwd en modo lectura */
fd_read = open("/etc/passwd", O_RDONLY);
if (fd_read < 0) {
    perror("open");
    exit(1);
}

/* Abre el fichero run.log (en el directorio actual) en modo escritura
   y lo trunca si tiene contenidos previos */
fd_write = open("run.log", O_WRONLY | O_TRUNC);
if (fd_write < 0) {
    perror("open");
    exit(1);
}
```

Llamadas al sistema para E/S

```
/* Abre el fichero /var/data/food.db en modo lectura y escritura */
fd_readwrite = open("/var/data/food.db", O_RDWR);
if (fd_readwrite < 0) {
    perror("open");
    exit(1);
}

/* Abre el fichero /var/log/messages en modo escritura para añadir datos
   a los contenidos existentes */
fd_append = open("/var/log/messages", O_WRONLY | O_APPEND);
if (fd_append < 0) {
    perror("open");
    exit(1);
}
```

Llamadas al sistema para E/S

```
int creat(const char *path, mode_t mode);
```

- Crea un fichero con unos permisos determinados
- Equivalente a **`open`**() con los flags

  `O_WRONLY | O_TRUNC | O_CREAT`

```
int fd = creat(file, 0644);
if (fd == -1)
   /* error */
```

Llamadas al sistema para E/S

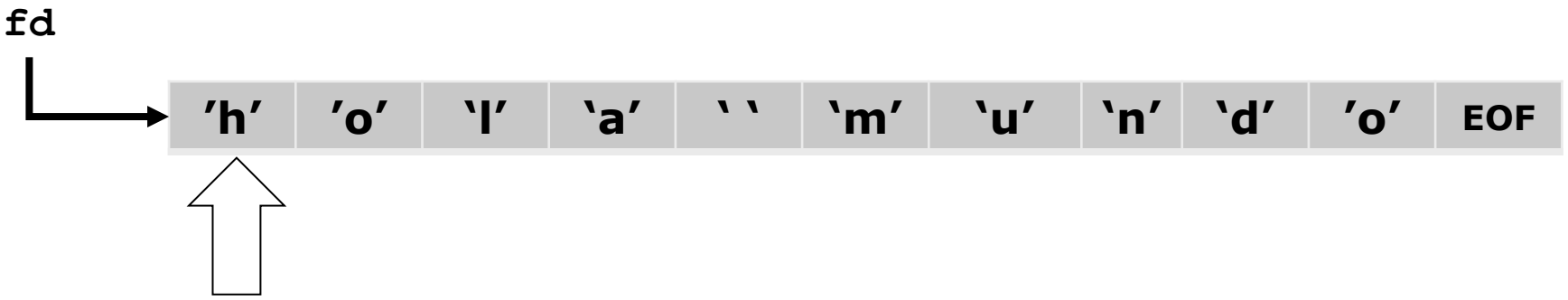
```
ssize_t read(int fd, void *buf, size_t nbytes);

ssize_t write(int fd, const void *buf, size_t nbytes);
```

- Valor devuelto:
  - Ambas funciones devuelven el **número de bytes realmente leídos o escritos** (pueden ser menos que lo indicado en el argumento nbytes)
  - **read() devuelve cero al final de fichero**: así detectamos el final de fichero
  - **-1** en caso de error

- El tipo **size_t** indica un tamaño (valor entero sin signo)
- El tipo **ssize_t** indica un tamaño (pero también admite valores negativos, con signo como "-1" para indicar errores)

Llamadas al sistema para E/S

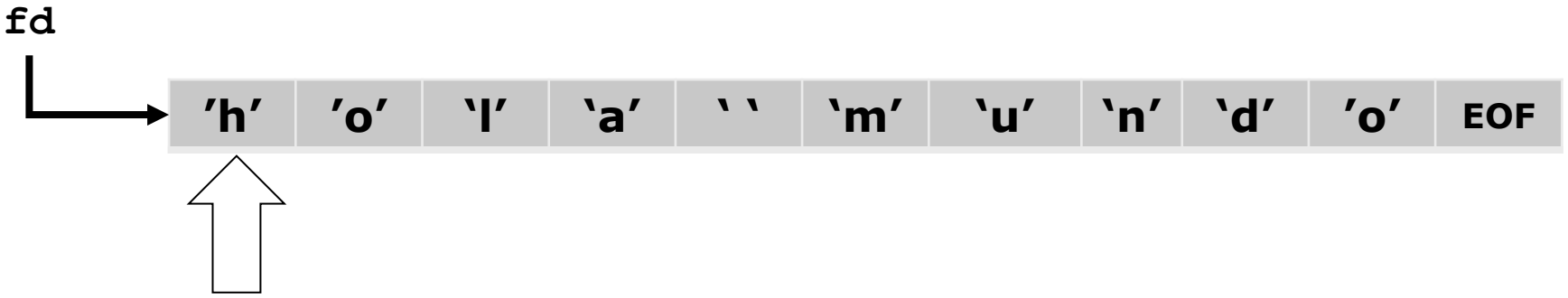



```
#define T 16
char b[T];
```

```
int fd = open("data.txt", O_RDONLY);
/* control de errores */

1.- read(fd, b, 4);
2.- read(fd, b, T);
3.- read(fd, b, 4);
    read(fd, b, 4);
```

Llamadas al sistema para E/S

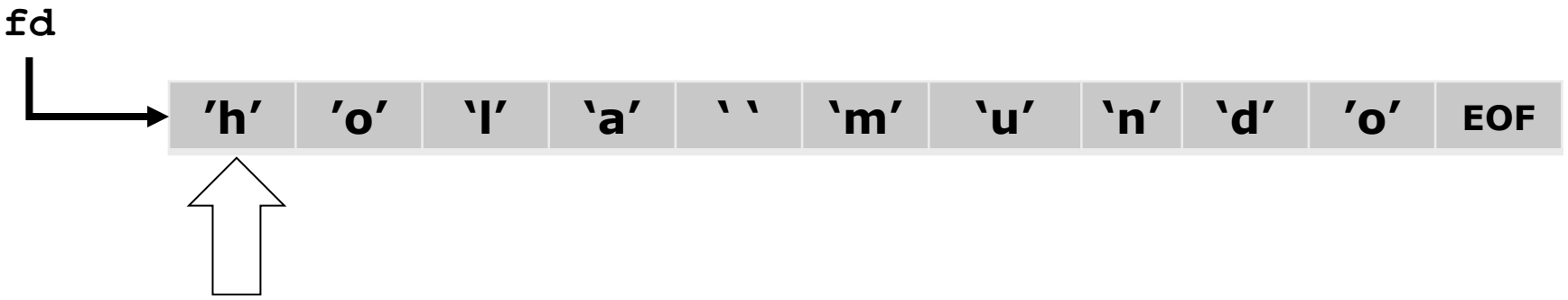



```
#define T 16
char b[T];

int fd = open("data.txt", O_WRONLY);
/* control de errores */

1.- write(fd, "123", 3);
```

Llamadas al sistema para E/S

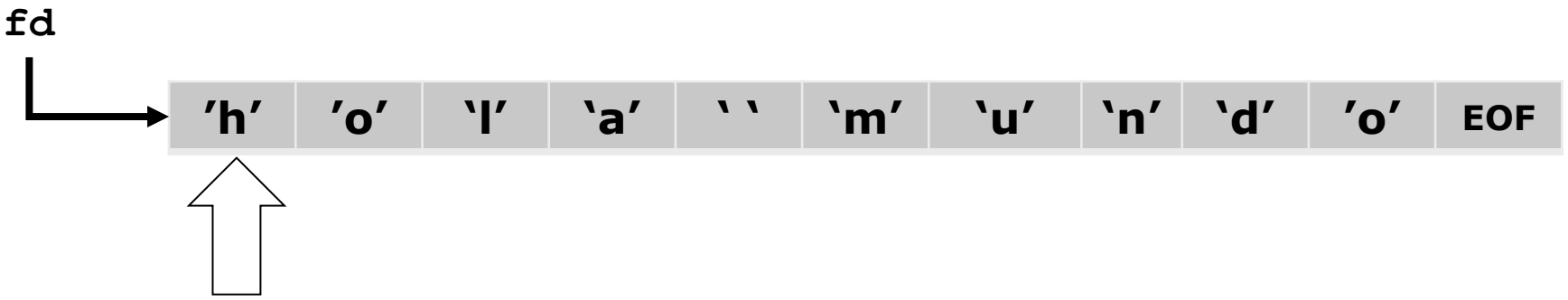



```
#define T 16
char b[T];

int fd = open("data.txt", O_WRONLY | O_TRUNC);
/* control de errores */

1.- write(fd, "123", 3);
```

Llamadas al sistema para E/S




```
#define T 16
char b[T];

int fd = open("data.txt", O_WRONLY | O_APPEND);
/* control de errores */

1.- write(fd, "123", 3);
```

Llamadas al sistema para E/S

```
/* guarda el resultado de read() */
ssize_t r;

/* buffer donde leer los datos         */
char buf[20];

/* lee 20 bytes del fichero            */
r = read(fd, buf, 20);
if (r == 0) {
    printf("Fin de fichero\n");
}
else if (r < 0) {
    perror("read");
    exit(1);
}
else {
    printf("leidos: %d bytes\n", r);
}
```

Llamadas al sistema para E/S

```
/* valor de retorno de write() */
ssize_t w;

/* escribe una cadena a un fichero */
w = write(fd, "hello world\n", strlen("hello world\n"));
if (w < 0) {
    perror("write");
    exit(1);
}
else {
    printf("Se escribieron %d bytes\n", w);
}
```

Llamadas al sistema para E/S

```
int close(int fildes);
```

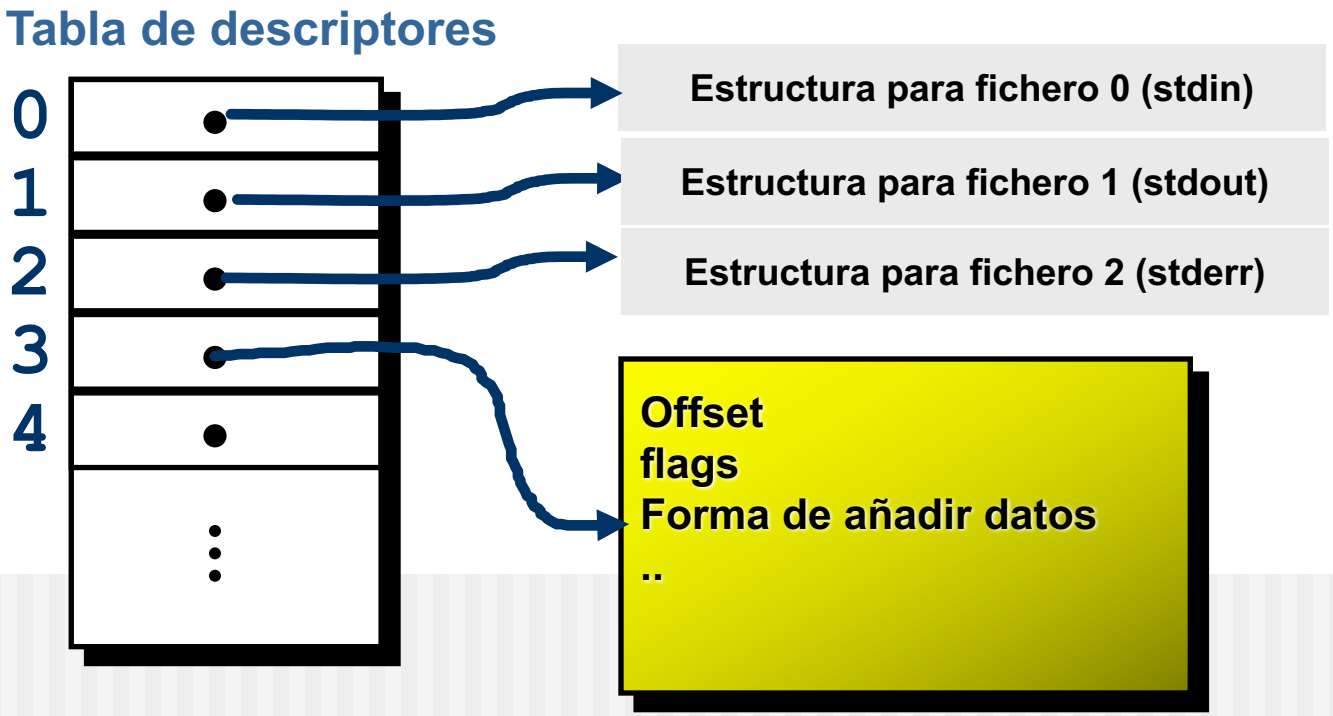
- Cierra un fichero
- NOTA: comprobar que no devuelve -1

```
if (close(fd) == -1) {
    perror("close");
    exit(1);
}
```

Entrada/Salida y Ficheros

- Un proceso en Linux puede disponer de 1024 descriptores de ficheros. Los tres primeros se abren al comenzar la ejecución (stdin, stdout, stderr)

- Hay dos formas de acceso para E/S: por llamadas al sistema o por funciones de librería C/C++



Información detallada de un fichero

```
int stat(const char *path, struct stat *buf);
```

- Devuelve información sobre el fichero en la estructura pasada por referencia
- Es útil para recorrer directorios
- struct stat está definida en <sys/stat.h>

Información detallada de un fichero (II)

Campos de la estructura stat:

```
mode_t   st_mode;     /* File mode (see mknod(2)) */
ino_t    st_ino;      /* Inode number */
dev_t    st_dev;      /* ID of device containing a directory entry for this file */
dev_t    st_rdev;     /* ID of device */
                      /* This entry is defined only for char special or block special files */
nlink_t  st_nlink;    /* Number of links */
uid_t    st_uid;      /* User ID of the file's owner */
gid_t    st_gid;      /* Group ID of the file's group */
off_t    st_size;     /* TAMAÑO DEL FICHERO EN BYTES */
time_t   st_atime;    /* Time of last access */
time_t   st_mtime;    /* Time of last data modification */
time_t   st_ctime;    /* Time of last file status change */
                      /* Times measured in seconds since */
                      /* 00:00:00 UTC, Jan. 1, 1970 */
long     st_blksize;  /* Preferred I/O block size */
blkcnt_t st_blocks;   /* Number of 512 byte blocks allocated*/
```

Funciones útiles para chequeo de valores de stat: **S_ISDIR(st_mode)**

S_ISREG(st_mode)

Información detallada de un fichero (III)

```
/* Estructura que se pasa a stat() para obtener resultados */
struct stat file_status;

/* Consulta la información de "foo.txt" y muestra la info por pantalla */

if (stat("foo.txt", &file_status) == 0) {
    if (S_ISDIR(file_status.st_mode))
        printf("foo.txt is a directory\n");
    if (S_ISLNK(file_status.st_mode))
        printf("foo.txt is a symbolic link\n");
    if (S_ISSOCK(file_status.st_mode))
        printf("foo.txt is a (Unix domain) socket file\n");
    if (S_ISREG(file_status.st_mode))
        printf("foo.txt is a normal file\n");
}
else { /* stat() ha devuelto -1 */
    perror("stat");
}
```

Estructura de directorios en Linux

- Algunos de los directorios principales en Linux:
  - /bin : Contiene los ejecutables (binarios) esenciales para el sistema. Si observamos su contenido encontraremos los comandos más básicos.
  - /boot : Aquí están los archivos usados por el sistema durante el arranque, incluida la imagen del núcleo.
  - /dev : Almacena los controladores (device drivers o device files) para el acceso a los dispositivos físicos del disco como el ratón , las tarjetas, el scanner, etc.
  - /var : Contiene información variable, tanto la generada por el propio sistema como por los usuarios.
  - /lib : Contiene librerías usadas por diferentes aplicaciones evitando que cada programa incluya las suyas propias; así se evita la redundancia.
  - /etc : Directorio usado para almacenar todos los archivos de configuración del sistema.
  - /proc: Directorio donde se encuentran ficheros de comunicación directa entre procesos de usuario y de kernel

Estructura de directorios en Linux (II)

- Algunos de los directorios principales en Linux:
  - /home : Contiene el árbol de directorios propios de cada usuario del sistema. Se encontrará un subdirectorio para cada usuario para salvaguardar la confidencialidad de sus datos. Es recomendable instalarlo en una partición diferente para salvaguardar los datos en caso de ocurrir una reinstalación del sistema.
  - /sbin : Aquí se encuentran los comandos esenciales de administración del sistema, normalmente reservados para el administrador (superusuario o root).
  - /usr : Aquí se almacenan las aplicaciones y recursos disponibles para todos los usuarios del sistema.
  - /tmp : Es el directorio temporal usado generalmente por las aplicaciones para almacenar algunos ficheros en tiempo de ejecución.
  - /media : Aquí descienden os ficheros dónde se montarán automáticamente las unidades extraíbles , como el cdrom , usb .
    - /media/cdrom0 : Para la primera unidad de cdrom.
    - /media/usbdisk : Para la memoria usb.
    - /media/floppy : Para el disquete.

Directorios en la librería de C

```
#include<stdio.h>
#include<sys/dir.h>
```

int mkdir(const char *path, mode_t mode);

- Crea un directorio con permisos

DIR *opendir(const char *dirname);

- opendir devuelve un puntero a estructura tipo DIR, para ser empleada en otras funciones

int closedir(DIR *dirp);

- cierre

Directorios en C

```
/* Abre el directorio "/home/users" para leer */
DIR* dir = opendir("/home/users");
if (!dir) {
    perror("opendir");
    exit(1);
}


if (closedir(dir) == -1) {
    perror("closedir");
    exit(1);
}
```

Directorios en C: recorrido de contenidos

struct dirent *readdir(DIR *dirp);

- Cada vez que se llama, se devuelve un fichero contenido en el directorio (diferente en cada llamada)
- Devuelve un puntero a una estructura de tipo dirent, que contiene, entre otros campos:
  - d_name[]: cadena con el nombre del fichero
  - d_namlen: longitud de la cadena anterior
- Devuelve NULL si se ha llegado al final del recorrido

```
struct dirent* entry;
printf("Contenidos del directorio:\n");
while ( (entry = readdir(dir)) != NULL) {
    printf("%s\n", entry->d_name);
}
```

- Se puede rebobinar hasta el principio para hacer otra pasada usando la función
  - rewinddir(DIR * dirp);