



Boletín 3- Archivos

Departamento de Lenguajes y
Sistemas Informáticos

Indice



1. Introducción al acceso a Archivos
 1. Acceso a archivos en alto nivel
 2. Acceso a archivos en bajo nivel
2. El acceso a los archivos en UNIX
3. El acceso a los directorios en UNIX

Introducción al acceso a Archivos

-Acceso mediante la Shell

\$echo “Hola Mundo” > HolaShell.txt

\$cp entrada.txt salida.txt

-Acceso a ficheros en alto nivel (FILE *)

FILE *fopen(const char *nom_archivo, const char *modo);

-Acceso a ficheros en bajo nivel (Descriptores de ficheros)

int open (char *name, int how_to_open, int mode);

Acceso a archivos en alto nivel

El acceso a los ficheros en C en alto nivel se basan en un tipo de datos de alto nivel **FILE***

Independiente del sistema operativo (ANSI C)

```
FILE *fopen(const char *nom_archivo, const char *modo);  
int fclose(FILE *fp);  
int putc(int car, FILE *pf);  
int getc(FILE *pf);  
fprintf(...), fscanf(...)
```

Acceso a archivos en alto nivel

Ejemplo: holaalto.c escribir una cadena en un archivo

```
#include <stdio.h>
char str[80]="Hola Mundo";

main()
{
    FILE *fp;
    char *p;

    fp = fopen ("HolaAlto.txt", "w");

    p = str;
    while (*p)
    {
        if (fputc (*p, fp) ==EOF)
        {
            printf("Error de escritura en el archivo \n");
            exit(1);
        }
        p++;
    }
    fclose(fp);
}
```

Acceso a archivos en bajo nivel

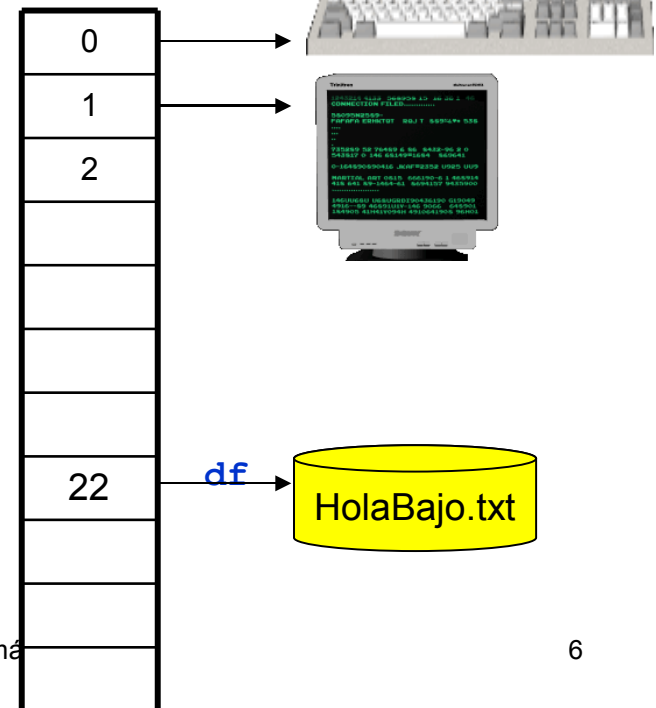
El sistema posee una tabla con los **descriptores de ficheros** de un programa. Existen ya predefinidos tres descriptores:

0	stdin (la entrada estandar)
1	stdout (la salida estandar)
2	stderr (la salida de errores)

El mecanismo de acceso a ficheros a bajo nivel:

- **abrir** el fichero asociándole un descriptor nuevo
- **acceder** al fichero identificado por el descriptor (leer/escribir)
- **cerrar** el fichero identificado por el descriptor

TABLA DE
DESCRIPTORES
DE FICHeros



Acceso a archivos en bajo nivel

Ejemplo: holabajo.c

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
main()
{
    int fd;
    char buf[255]="Hola Mundo";

    fd = open("HolaBajo.txt", O_CREAT|O_WRONLY, 0744);
    write(fd, buf, strlen(buf));
    close(fd);
}
```

Acceso a ficheros a bajo nivel

Apertura de ficheros `open()`

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

int open (char *name, int how_to_open);
int open (char *name, int how_to_open, int mode);
```

Parámetros

name es el nombre de fichero.

how_to_open indica la forma en que se abre (*)

mode especifica los permisos de acceso al fichero

Devuelve

int el descriptor de ese fichero. En caso de error -1

```
fd= open ("mifich.dat", O_RDWR | O_CREAT | O_TRUNC, 0700);
```


Acceso a ficheros a bajo nivel

Apertura de ficheros **open()**

how_to_open indica la forma en que se abre (*)

<fcntl.h>

- O_RDONLY:** Abre el fichero sólo para leer.
- O_WRONLY:** Abre el fichero sólo para escribir
- O_RDWR:** Abre el fichero para leer y escribir
- O_APPEND:** Añade al fichero si se escribe, en vez de truncar.
- O_TRUNC:** Trunca el fichero (lo deja con 0 bytes) si se abre para escribir.
- O_CREAT:** Crea el fichero si no existe. El tercer parámetro indicarán los permisos de acceso al fichero.
- O_EXCL:** Produce error si el fichero se intenta crear pero ya existe.
- O_NDELAY:** No bloquea el proceso al acceder al fichero. No nos detendremos en este aspecto.

Acceso a ficheros a bajo nivel

Lectura/Escritura de ficheros `read()`, `write()`

```
#include <sys/file.h>
#include <unistd.h>
int read (int fd, void *buffer, int num_bytes);
int write (int fd, void *buffer, int num_bytes);
```

Parámetros

fd es el descriptor de fichero obtenido con `open()`
También se pueden usar los descriptores que ya están definidos (0,1,2)

***buffer** puntero al buffer que se lee o escribe

num_bytes número de bytes que se leen/escriben

Devuelve

int el número de bytes leídos/escritos. En caso de error -1.
En el caso de `read`, devuelve 0 al llegar al EOF

Acceso a ficheros a bajo nivel

Cierre de ficheros **close()**

```
#include <sys/file.h>  
int close (int fd);
```

Parámetros

fd es el descriptor de fichero.

Devuelve

int **0 en caso de cierre correcto.** En caso de error -1

Acceso a ficheros a bajo nivel

Ejemplo: copia.c

```
main()
{
    int fd_origen, fd_destino;
    char c;
    fd_origen = open("entrada.txt", O_RDONLY);
    fd_destino = open("salida.txt", O_CREAT|O_WRONLY, 0700);
    while ( read(fd_origen, &c, sizeof(char)) > 0 )
    {
        write(fd_destino, &c, sizeof(char));
    }
    close(fd_origen);
    close(fd_destino);
}
```

Acceso a archivos a bajo nivel

Paso de bajo nivel a alto nivel **fdopen()**

```
#include <stdio.h>
```

```
FILE *fdopen (int fd, char *mode);
```

Parámetros

fd es el descriptor de fichero obtenido con `open`.

mode especifica los permisos de acceso al fichero.

Devuelve

FILE un puntero a la estructura **FILE**. NULL caso de error

```
FILE *pf;  
int fd_origen;  
fd_origen = open("entrada.txt", O_RDONLY);  
pf = fdopen(fd_origen, "r");  
fprintf(pf, "hola mundo");
```

Acceso a directorios en UNIX

getcwd() – Acceso al directorio de trabajo

```
#include <unistd.h>
char *getcwd(char *buf, size_t size);
```

Parámetros

buf es la variable dónde se almacenará el nombre del directorio actual de trabajo.

size es el tamaño en bytes de **buf**.

Devuelve

char * **NULL** en caso de error y un puntero a la zona de memoria donde se almacena el nombre del directorio de trabajo actual

Acceso a las entradas de un directorio

opendir(), readdir(), rewinddir(), closedir()

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
struct dirent *readdir(DIR *dir);
void rewinddir(DIR *dir);
int closedir(DIR *dir);
```

Las funciones referencian al directorio con la estructura de datos DIR (Identificador de un directorio)

Acceso a las entradas de un directorio

opendir() – devuelve un identificador de directorio

```
#include <sys/types.h>
#include <dirent.h>
```

```
DIR *opendir(const char *nombre_dir);
```

Parámetros

nombre_dir nombre del directorio

Devuelve

DIR * puntero a un tipo de datos que referencia al directorio o NULL en caso de error

Acceso a las entradas de un directorio

readdir() – lectura de las entradas de un directorio

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir(DIR *dir);
```

Parámetros

dir puntero a la variable de directorio obtenida en opendir()

Devuelve

struct dirent * puntero a la estructura de datos con la información de la entrada de directorios o NULL en el caso de que no haya más entradas.

Se pueden hacer tantas operaciones readdir() como entradas haya en un directorio

Acceso a las entradas de un directorio

La estructura de datos dirent

```
struct dirent {  
  
    ino_t          d_ino;      /* Numero de i-nodo */  
  
    off_t          d_off;      /* Offset de la siguiente entrada del directorio */  
  
    unsigned short d_reclen;    /* Longitud de la estructura de la entrada de directorio */  
  
    char           d_name[MAXNAMELEN]; /* Nombre del archivo */  
  
};
```

```
struct dirent *direntp;  
direntp = readdir( dirp );  
printf("%s\n", direntp->d_name );
```

Acceso a las entradas de un directorio

Ejemplo: imprimir las entradas de un directorio

```
#include <dirent.h>
#include <sys/types.h>
int main()
{
    DIR *dirp;
    struct dirent *direntp;
    if ((dirp = opendir("/etc")) == NULL)
        printf("No se puede abrir el directorio\n");
    while ( (direntp = readdir( dirp )) != NULL )
        printf("%s\n", direntp->d_name );
    closedir(dirp);
}
```

Información de una entrada directorio

Las funciones stat(), fstat()

```
#include <sys/stat.h>
#include <unistd.h>
/* Usando un nombre de fichero) */
int stat(const char *file_name, struct stat *buf);
/* Usando un descriptor de fichero) */
int fstat(int filedes, struct stat *buf);
/* Para ficheros especiales que son enlaces */
int lstat(const char *file_name, struct stat *buf);
```

Devuelve

int la función stat devuelve 0 en caso correcto y -1 en caso de error

Información de una entrada directorio

La estructura de datos stat

```
struct stat{
    dev_t          st_dev;          /* dispositivo */
    ino_t          st_ino;          /* inodo */
    mode_t         st_mode;         /* protección */
    nlink_t        st_nlink;        /* número de enlaces físicos */
    uid_t          st_uid;          /* ID del usuario propietario */
    gid_t          st_gid;          /* ID del grupo propietario */
    dev_t          st_rdev;         /* tipo dispositivo inodo */
    off_t          st_size;         /* tamaño total, en bytes */
    unsigned long  st_blksize;      /* tamaño de bloque para el
                                   sistema de ficheros de E/S */
    unsigned long  st_blocks;       /* número bloques asignados */
    time_t         st_atime;        /* hora último acceso */
    time_t         st_mtime;        /* hora última modificación */
    time_t         st_ctime;        /* hora de creación */
};
```

Información de entrada de directorio

La estructura de datos **stat**

Valores posibles del campo **st_mode**

<sys/stat.h> define constantes que hace más fácil la comprobación

S_IFREG	0100000	Es fichero regular
S_IFDIR	0040000	Es directorio
S_IFSOCK	0140000	Es un socket
S_IFLNK	0120000	Es enlace simbólico
S_IFBLK	0060000	Es dispositivo de bloques
S_IFCHR	0020000	Es dispositivo de caracteres
S_IFIFO	0010000	Es fifo o tubería nombrada

Información de entrada de directorio

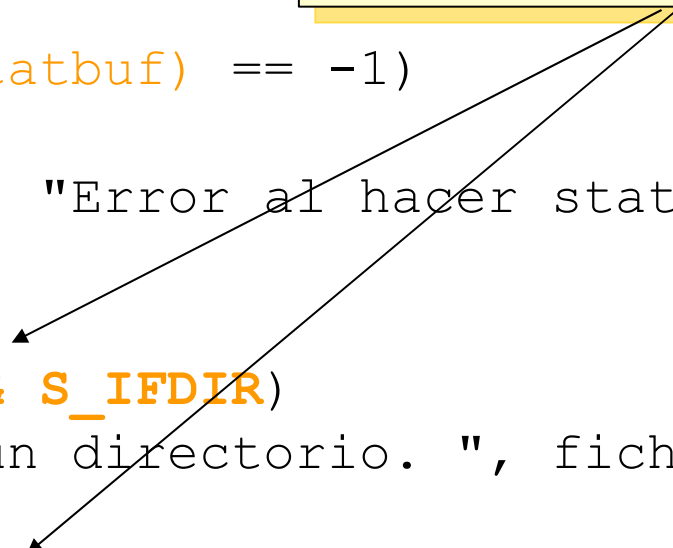
Ejemplo: información de un archivo usando `stat` y `st_mode`

```
struct stat statbuf;
strcpy(fichero, "prueba.txt");

if (stat(fichero, &statbuf) == -1)
{
    fprintf(stderr, "Error al hacer stat \n");
    exit(1);
}
if (statbuf.st_mode & S_IFDIR)
    printf("%s es un directorio. ", fichero);

if (statbuf.st_mode & S_IFREG)
    printf("%s es un fichero. ", fichero);
```

SE USA **AND** BINARIO PARA
OBTENER LA INFORMACIÓN



Acceso a ficheros a bajo nivel

CONSIDERACIONES DURANTE LA PROGRAMACIÓN DE ESTE BOLETÍN

- En el tratamiento de **directorios**:
- Definir los punteros a los directorios **DIR *idDir** y a las entradas de directorios **struct dirent *ent**
- Control de errores en **opendir()** --> devuelve **NULL**
- Control de errores en **readdir()** --> devuelve **NULL** cuando error o EOF
- Tratar los campos de la estructura como campos de un puntero a struct (**ent->d_name**)
- Cerrar la lectura **closedir()**

Acceso a ficheros a bajo nivel

- **En el tratamiento de archivos:**
- Definir la estructura de datos **struct stat statbufer** (ojo que no es un puntero).
- Paso de la variable statbufer por referencia a la función **stat("fichero",&statbufer)**
- Comparaciones en binario para la variable **st_mode** de statbufer (**statbufer->st_mode & IF_DIR**)
- Definir los **int** descriptores de ficheros.
- En el **open()** controlar los errores --> devuelve **-1**
- Modos de acceso en **open()** con el operador OR → **O_RDWR|O_CREAT|O_TRUNC**
- En el **read()/write()** pasar por referencia la variable a leer **read(df,&buffer,tam)** y controlar los errores --> devuelve **0 en caso de EOF** y **-1 en caso de error**
- Realizar el **close()** del fichero
- No realizar **open()** cuando haya que leer o escribir en la entrada/salida estandar