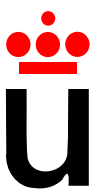
Tema 3: Ficheros

Enrique Soriano

Laboratorio de Sistemas,
Grupo de Sistemas y Comunicaciones,
URJC

3 de marzo de 2010



Universidad
Rey Juan Carlos

(cc) 2010 Grupo de Sistemas y Comunicaciones.

Algunos derechos reservados. Este trabajo se entrega bajo la licencia Creative Commons Reconocimiento - NoComercial - SinObraDerivada (by-nc-nd). Para obtener la licencia completa, véase http://creativecommons.org/licenses/by-sa/2.1/es. También puede solicitarse a Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

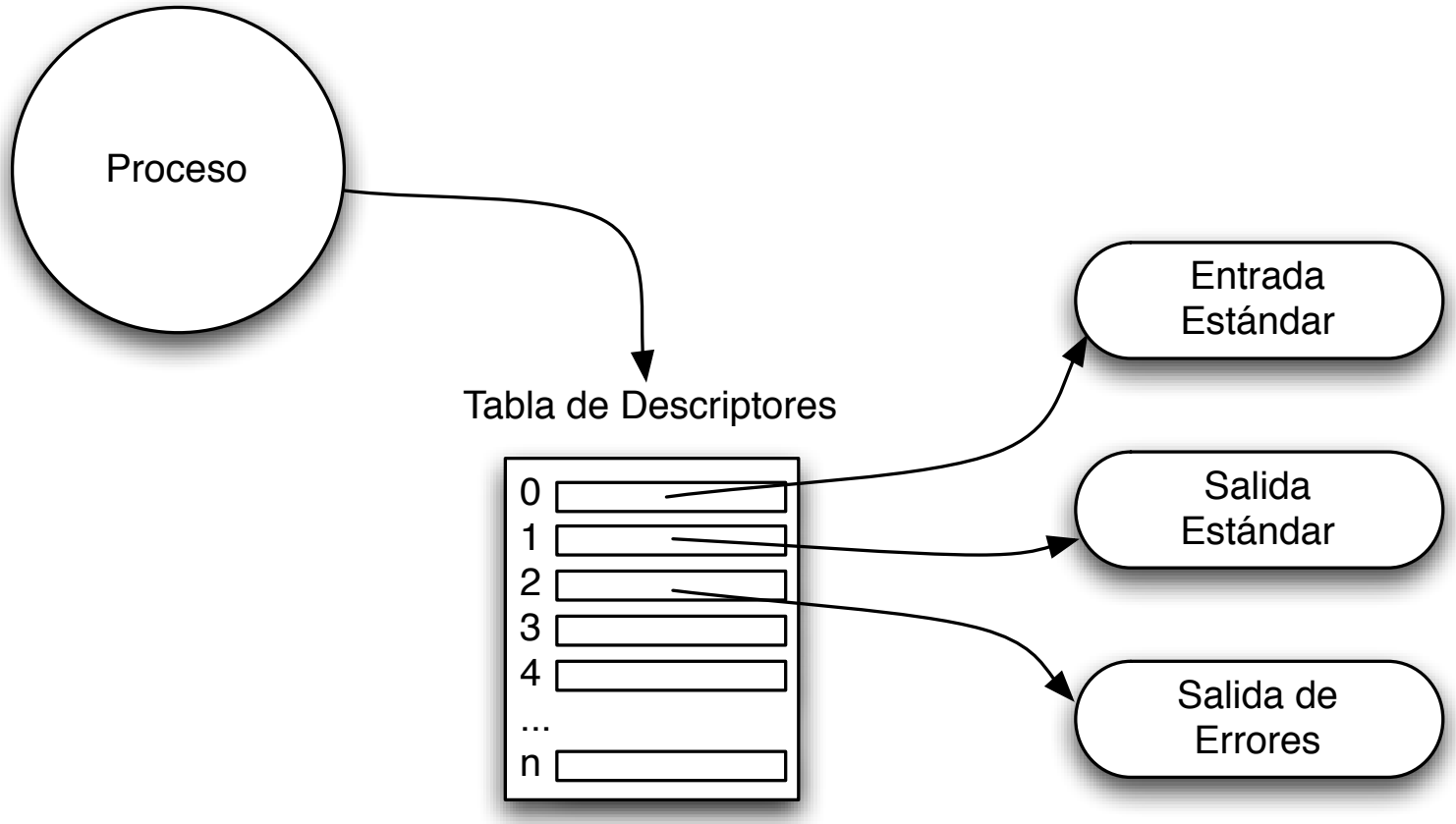
Uso de ficheros

- Ya hemos usado ficheros desde nuestros programas, p. ej. `print()`.
- Operaciones básicas: open, read, write, close.

Descriptores de fichero

- Cada fichero que está siendo usado por un proceso tiene al menos un descriptor de fichero (**file descriptor**) que lo identifica.
- El descriptor de fichero es un número entero que se usa como índice en la **tabla de descriptores de fichero**.
- Hay tres posiciones especiales: (0) entrada estándar, (1) salida estándar, y (2) salida de errores.

Descriptores de fichero



Read

- `long read(int fd, void *buf, long nbytes);`
- Lee como mucho nbytes del fichero abierto en ese descriptor.
- Puede devolver menos sin ser error.
- Si devuelve 0 bytes, se ha llegado al final del fichero.

Write

- `long write(int fd, void *buf, long nbytes);`
- Escribe nbytes en el fichero abierto en ese descriptor.
- Si devuelve distinto a nbytes, se debe considerar un error.

ver rwstdio.c

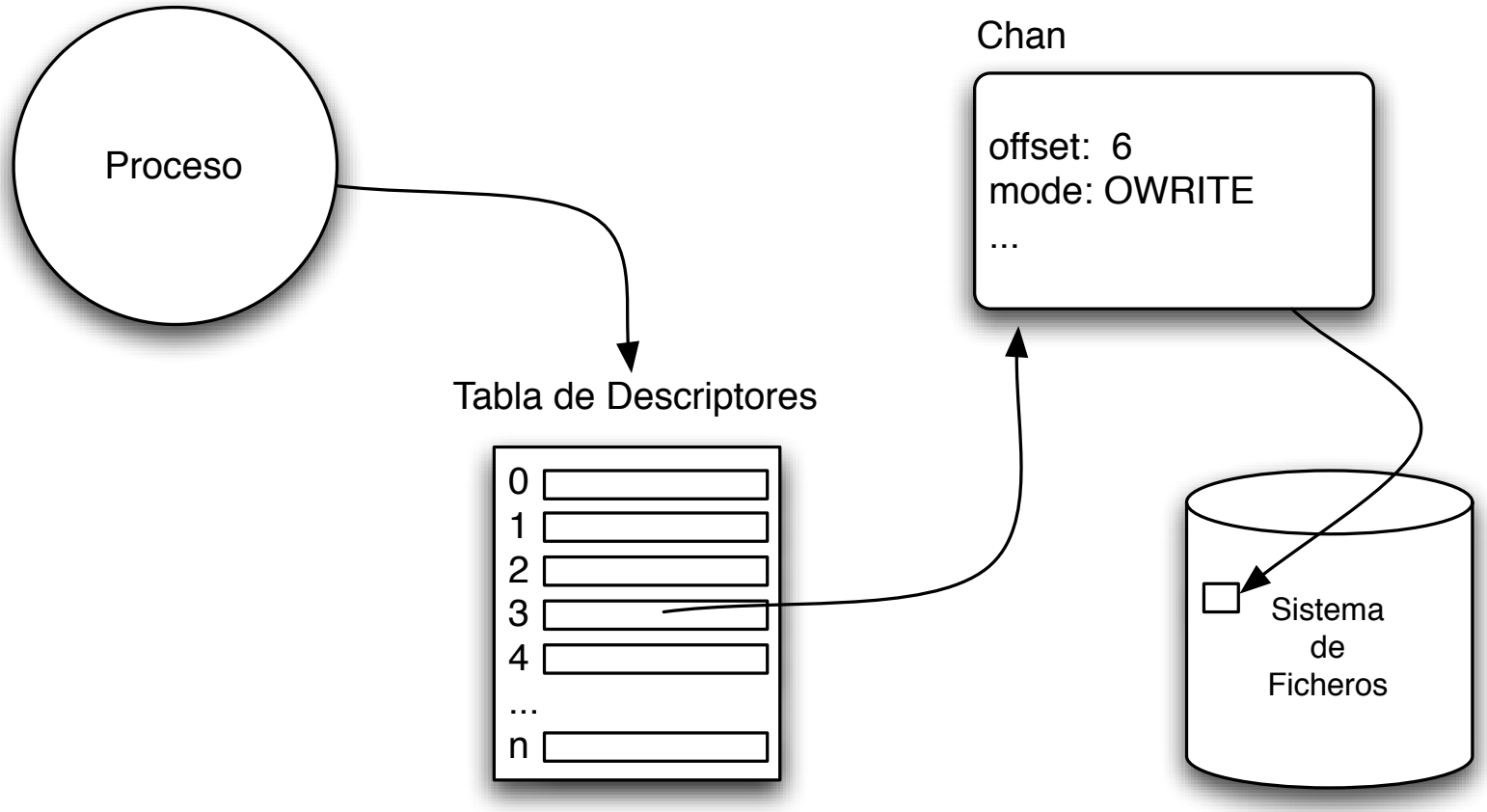
Open

- `int open(char *file, int omode);`
- Bits de modo: `OREAD`, `OWRITE`, `ORDWR`.
- Usa el primer descriptor de fichero libre.
- Cuando se abre un fichero, el sistema mantiene el **offset**, que se actualiza cada vez que se lee o se escribe en él. Empieza a 0.
- El bit de modo `OTRUNC` trunca el fichero, lo deja a 0 bytes.

Close

- `int close(int fd);`
- Hay que cerrar el descriptor cuando ya no se va a volver a usar.

Chans



ver write.c y trunc.c

Seek

- `vlong seek(int fd, vlong n, int type);`
- Si `type` es 0 el offset se pone a `n`.
- Si `type` es 1 se suma `n` al offset actual.
- Si `type` es 2 se suma `n` a la última posición del fichero.

Create

- `int create(char* file, int mode, ulong perm);`
- Bits para `perm`: DMDIR, DMAPPEND, DMEXCL
- Los permisos del fichero dependen de `perm` y los permisos del directorio en el que se crea.
- Si el fichero existe, se trunca a 0.

ver huecos.c

Remove

- `int remove(char* file);`
- No se puede borrar un directorio con contenido (como con `rm`).

Access

- `int access(char* file, int mode);`
- Devuelve cero si el fichero puede accederse en el modo indicado, -1 en otro caso.
- Bits para `amode`: AEXIST, AWRITE, AREAD, AEXEC.

Entradas de directorio

- Un directorio es un *fichero* con metadatos o atributos de otros ficheros.
- Directorio vacío $\longrightarrow$ fichero vacío.
- Directorio con entradas $\longrightarrow$ fichero con entradas de directorio.
- Las entradas de directorio tienen un formato independiente de arquitectura.
- No se pueden escribir los directorios con write(), es peligroso.

Dir

- `type,dev,qid`: identifican al fichero en el FS.
- `mode`: permisos y tipo de fichero.
- `atime`: última fecha de lectura, secs. desde epoch.
- `mtime`: última fecha de escritura, secs. desde epoch.
- `length`: tamaño de fichero, en bytes. 0 para directorios.
- `name`: última componente de la ruta (string).
- `uid`: dueño (string).
- `gid`: grupo (string).
- `muid`: usuario de última modificación (string).

Dirstat

- `Dir* dirstat(char* file);`
- `Dir* dirfstat(int fd);`
- Devuelve una estructura Dir en memoria reservada con malloc.
- Hay que liberar el puntero después de usarlo.

ver size.c

Dirwstat

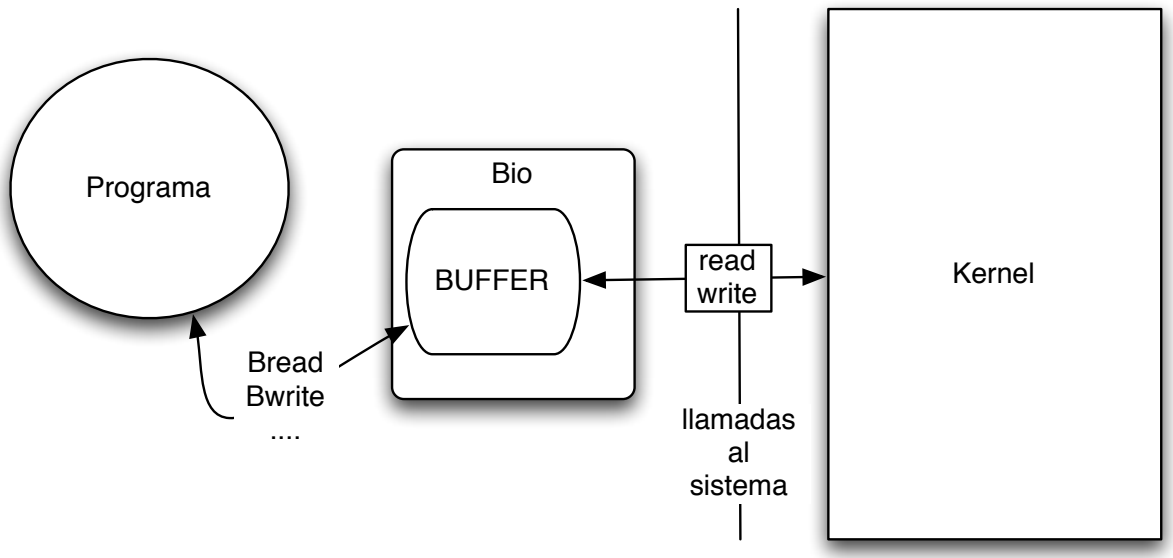
- `int dirwstat(char* file, Dir *dir);`
- `int dirfwstat(int fd, Dir *dir);`
- `void nulldir(Dir *dir);`
- Sólo se pueden cambiar algunos atributos: `nombre`, `modo`, `mtime`, `tamaño`, `gid`, `uid` (depende del FS).
- Atómico: o se cambian todos los requeridos, o ninguno. -1 si error.

Dirread

- `long dirread(int fd, Dir **buf);`
- Deja en el puntero que se le pasa por referencia un array de estructuras Dir con las entradas del directorio.
- Devuelve el número de entradas leídas. Hay que leer hasta que devuelve 0.
- Hay que liberar el puntero después de usarlo.

Bio

- A veces no es fácil leer de un fichero: p. ej. leer líneas de un fichero.
- Realizar una llamada al sistema sale caro: p. ej. leer carácter a carácter.
- Bio se interpone y te ofrece *buffering*.



Interfaz de Bio

- `Biobuf * Bopen(char *file, int mode);`
- `int Binit(Biobufhdr *bp, int fd, int mode);`
- `int Bterm(Biobufhdr *bp);`
- `long Bread(Biobufhdr *bp, void *buf, long nbytes);`
- `long Bwrite(Biobufhdr *bp, void *buf, long nbytes);`
- `void * Brdline(Biobufhdr *bp, int delim);`
- `int Blinelen(Biobufhdr *bp);`
- `char * Brdstr(Biobufhdr *bp, int delim, int nulldelim);`
- `int Bflush(Biobufhdr *bp);`

ver bio.c

Shell Globbing

- Para trabajar con ficheros de forma cómoda **en el shell**, usamos caracteres comodín.
- Para *escapar* estos caracteres, podemos usar comillas simples.
- El shell expande si puede, si no lo pasa literalmente.

```
*   ?   [ ]   [~ ]
```

```
ls *.txt
ls notas200?.txt
echo [a-m]*
rm */*/*.c
cat notas200[56].txt
cat notas200[~0-47-9].txt
```