TEMA 3. Sistema de Ficheros y E/S

Competencia C3: Usar las funciones de E/S de un Sistema Operativo para programar utilidades.

Contenidos

1. Introducción a concepto de Independencia del Dispositivo, redirecciones.
2. Llamadas al sistema de E/S y funciones de biblioteca estándar de C.
3. Modos de operación sobre dispositivos y ficheros
4. Organización del Sistema de ficheros. Espacios de nombres, directorios.
5. "Buffering" de E/S: En las bibliotecas de E/S y en las llamadas al sistema.
6. Acceso avanzado a las propiedades de los dispositivos.
7. Ejercicios y proyecto de E/S.

Blibliografía

[Capítulos 2,3 y 4] F.M. Márquez: UNIX. Programación Avanzada 3ª Edición. Rama, 2004.

[Apartados 5.2 y 5.3] C. Rodríguez, I. Alegria, J. González, A. Lafuente: *Descripción Funcional de los Sistemas Operativos*. Síntesis, 1994

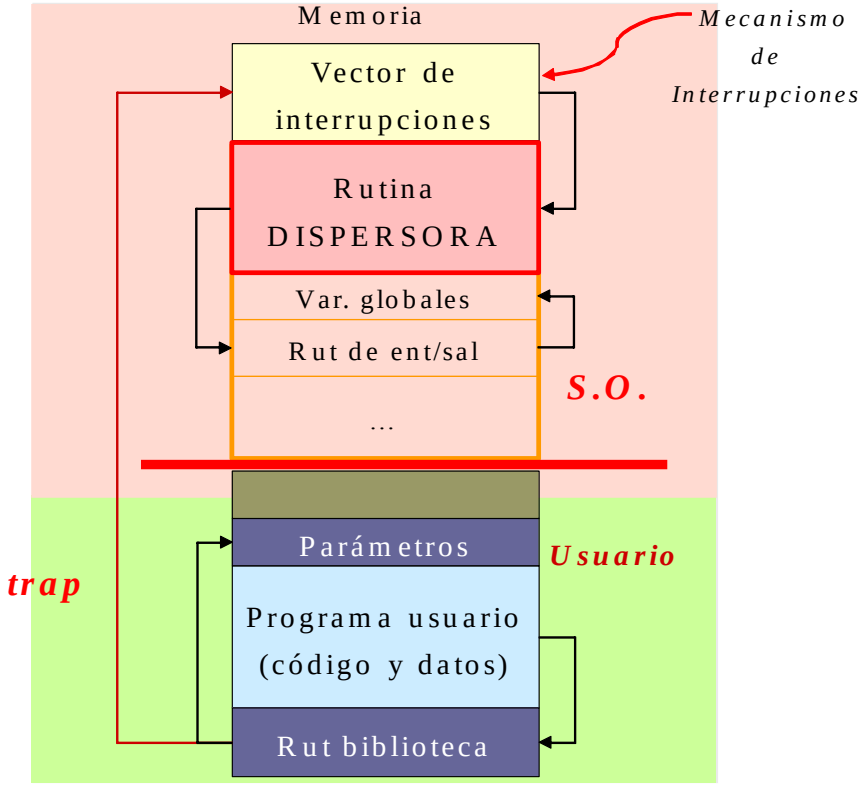
[Capítulo 12] W. Stallings: Sistemas Operativos. 5º Ed. Pearson Prentice-Hall, 2005.

[Capítulo 13] G. Nutt: Sistemas Operativos. 3º Ed. Pearson Addison Wesley, 2001

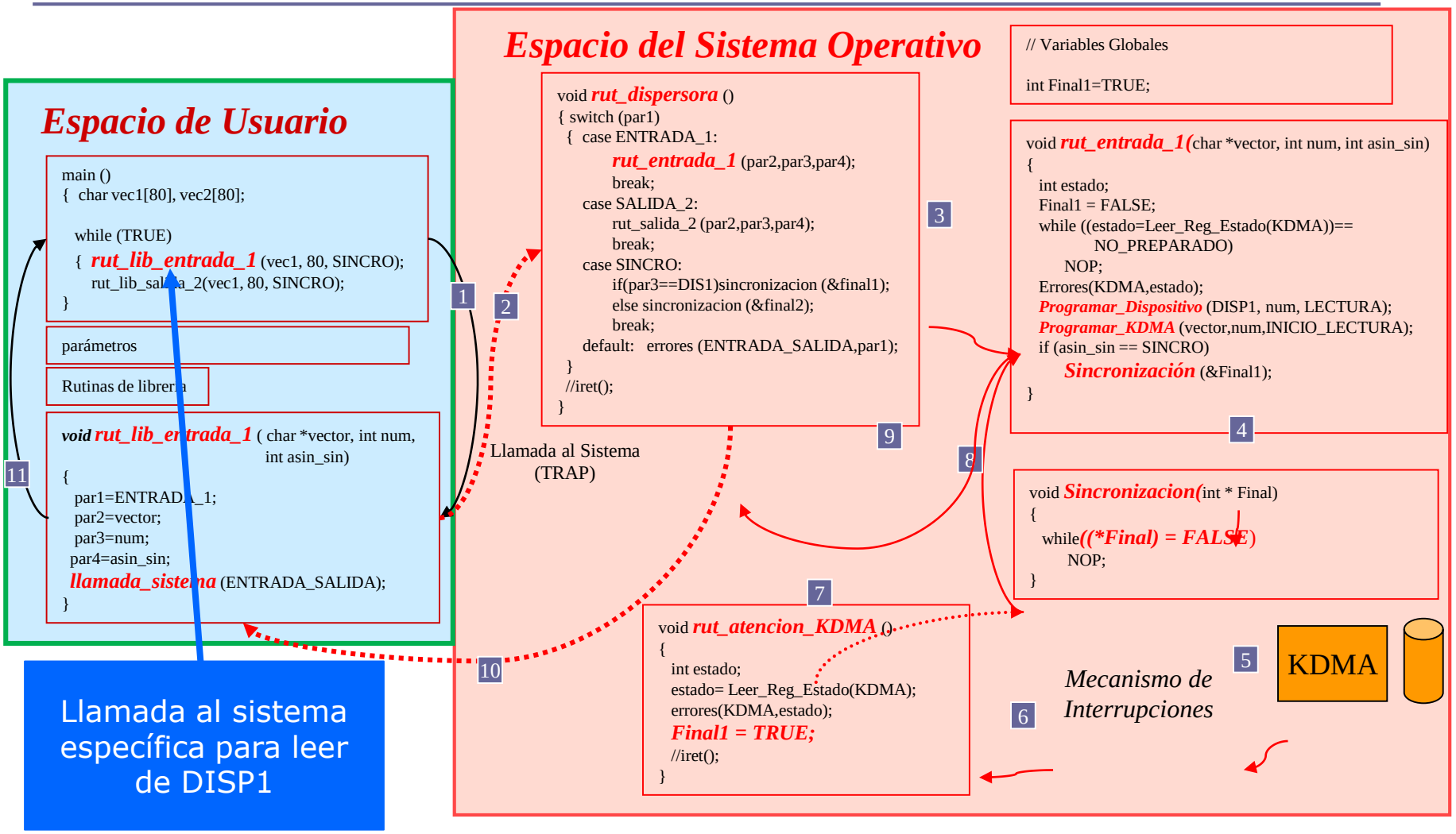
[Capítulo 12] A. Silberschartz: Operating Systemas Concepts. Sixth, John Wiley & Son, 2003

3.1 Introducción a concepto de Independencia del Dispositivo, redirecciones.

a) Acceso al sistema operativo mediante llamadas al sistema



Acceso al S.O. operativo mediante llamadas al sistema. Ejemplo estudiado:



Independencia del Dispositivo

Problema: *¿Cómo programar aplicaciones sin pensar en qué dispositivos de entrada/salida usan?*

☞ Según el esquema visto en el tema 2:

- Un dispositivo ⇨ Una llamada al sistema específica
- Cambiar de dispositivo ⇨ Modificar el código del programa

¿Cómo independizar el programa de los dispositivos utilizados?

Solución: *uso de dispositivos lógicos*, ***canales***

- El **canal** como abstracción del dispositivo
- En UNIX: ***file descriptor***

Concepto de ***independencia de los dispositivos***

Independencia del dispositivo

```
void rut_lib_leer (int canal char *vector,
                   int num, int asin_sin)
{
  par0=canal;
  par1=LEER;
  par2=vector;
  par3=num;
  par4=asin_sin;
  llamada_sistema (ENTRADA_SALIDA);
}
```

```
void rut_lib_escribir (int canal char *vector,
                       int num, int asin_sin)
{
  par0=canal;
  par1=ESCRIBIR;
  par2=vector;
  par3=num;
  par4=asin_sin;
  llamada_sistema (ENTRADA_SALIDA);
}
```

Programa usuario (síncrono)

```
main ()
{
  int f1, f2;
  char vec1[80], vec2[80];
  rut_lib_abrir(f1,"dispositivo_entrada" );
  rut_lib_abrir(f2,"dispositivo_salida");
  while (TRUE) {
     rut_lib_leer (f1,vec1,80,SINCRO);
     rut_lib_escribir (f2,vec1,80,SINCRO);
   }
  rut_lib_cerrar (f1);
  rut_lib_cerrar (f2);
}
```

Canales preestablecidos (estándar)

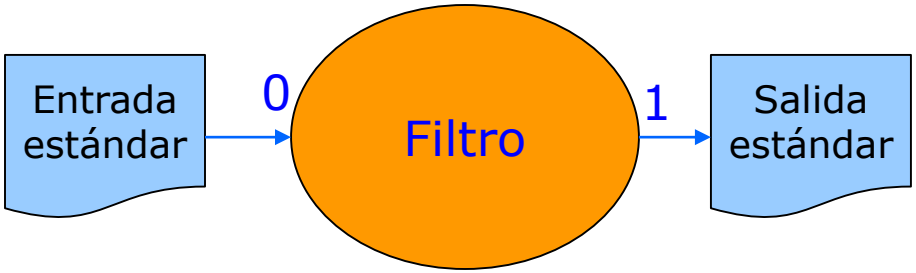
- Canales estándares = dispositivos preestablecidos (entrada, salida, error). En Unix/Linux: 0, 1, 2

  (*Biblioteca estándar de C:* ***stdin***, ***stdout***, *stderr*

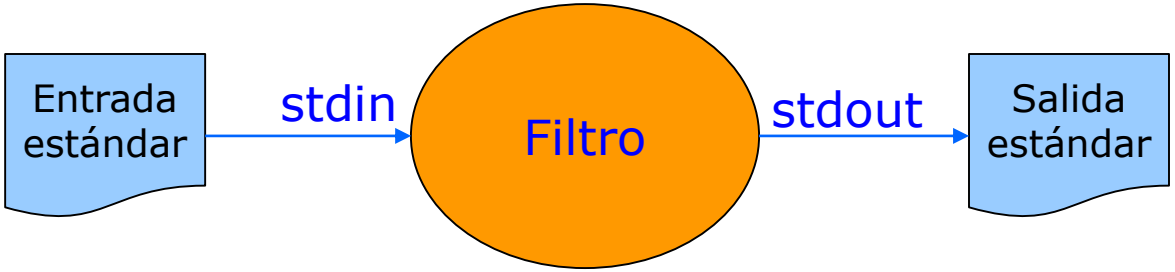
- El Interprete de Comandos (IC) dispone de mecanismos para trabajar con canales: (**<** **>** **2>**) (**>>**) Cuidado con (<<)
  - **Redirección de salida : [n]>word**
    - ls > listado.txt
    - ls 1> listado.txt
  - **Redirección de entrada : [n]<word**
    - cat < listado.txt
    - cat 0< listado.txt
  - **Redirección de salida añadiendo : [n]>>word**
    - cat >> listado.txt
  - **Redirección de salida de errores : 2>word**
  - **Redirección de salida estándar y salida de errores : &>word** o >&word

Concepto de proceso *"filtro"*

Llamadas al sistema



Biblioteca estándar de C



3.2 Llamadas al sistema de E/S y funciones de biblioteca estándar de C

- Llamadas al sistema de lectura y escritura

```
int read (int fd, void *buf, unsigned lon);
int write ( int fd, void *buf, unsigned lon);
```

- Funciones de biblioteca estándar C

  usando los canales estándar (implícitamente)

```
int getchar();
int putchar (int c);
int printf (const char *format, ...);
int scanf (const char *format, ...);
…
```

Act 04.2 Ejemplo de ccopiar.c con *funciones de biblioteca estándar de C*

```c
/**
* @file ccopiar.c
* @author G.A.
* @date 21 Jan 2018
* @brief Simplified version of cat command (without arguments)
*
*/

#include <stdio.h>

int main(int argc, char *argv[])
{
    int c;

    while ((c=getchar()) != EOF)
        putchar(c);

    return 0;
}
```

stdin (implícito)

stdout (implícito)

Act 04.3 Ejemplo de scopiar.c con *llamadas al sistema*

```c
/**
 * @file scopiar.c
 * @author G.A.
 * @date 21 Jan 2018
 * @brief Simplified version of cat command (without arguments).
 *
 */

#include <unistd.h>
#define TAMANO_BUFFER 512

int main(int argc, char *argv[])
{
    int n;
    char buf[TAMANO_BUFFER];

    while ((n= read(0, buf, TAMANO_BUFFER)) > 0)
        write(1, buf, n);

    return 0;
}
```

Canal 0 (explícito)

Canal 1 (explícito)

Actividad A04.4. Manejando mensajes de error

Canal estándar para la los mensajes de error (salida estándar de errores)

canal 2: con llamadas al sistema

stdout : con funciones de biblioteca estándar de C

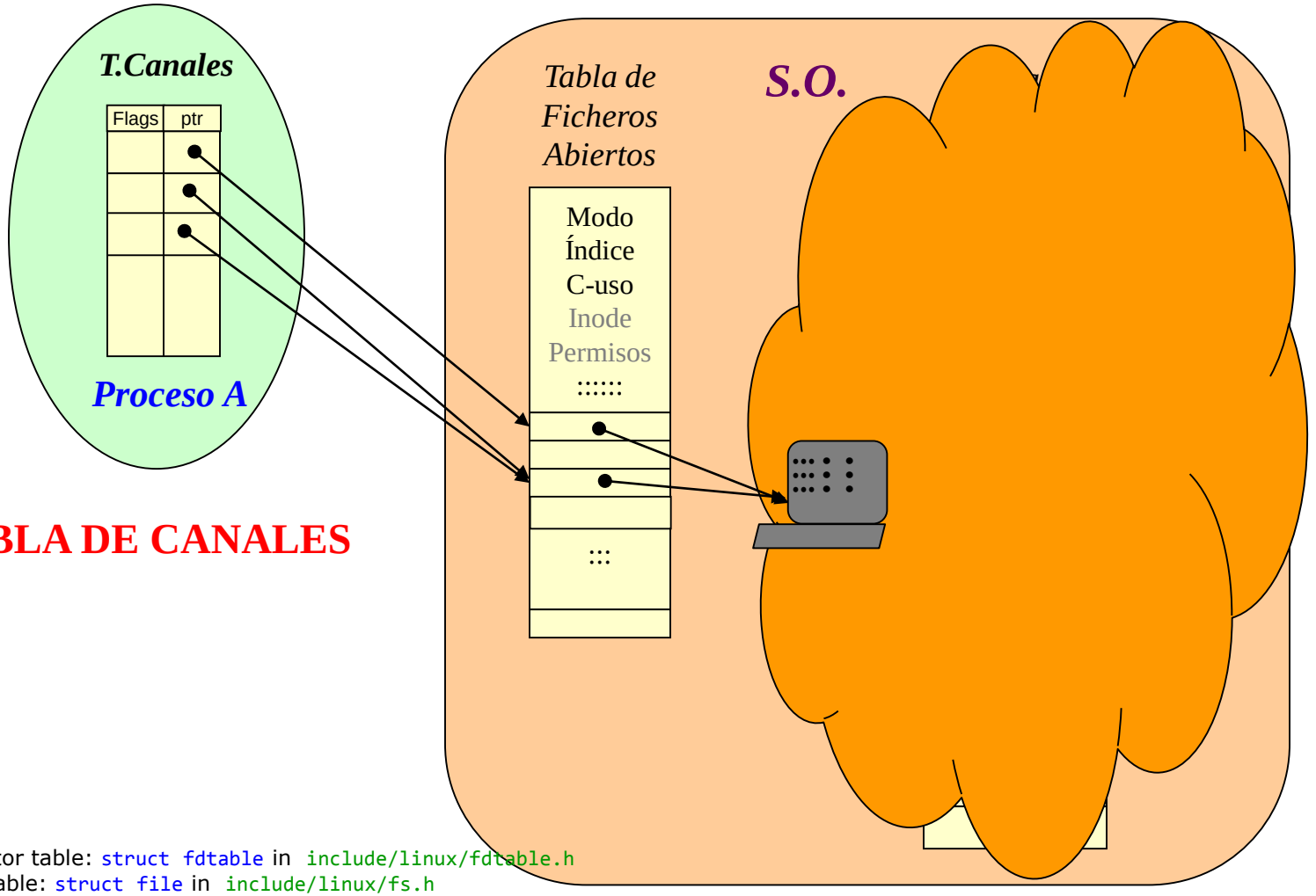
- **Redirección de salida de errores :**
  programa 2> word

Utilizando canales de entrada-salida adicionales a los estándar.

- Para comprender mejor la funcionalidad de las funciones adicionales de E/S con los canales necesitamos conocer cómo gestiona UNIX las operaciones de E/S.

- Los mecanismos que vamos a estudiar, son una simplificación de las estructuras reales que utiliza UNIX, pero nos sirven para intuir cómo las gestiona.

Independencia del Dispositivo en UNIX



(TC) File descriptor table: `struct fdtable` in `include/linux/fdtable.h`
(TFA)Open file Table: `struct file` in `include/linux/fs.h`
(TI) Inode table(In memory) : `struct inode` in `include/linux/fs.h`

Independencia del Dispositivo en UNIX

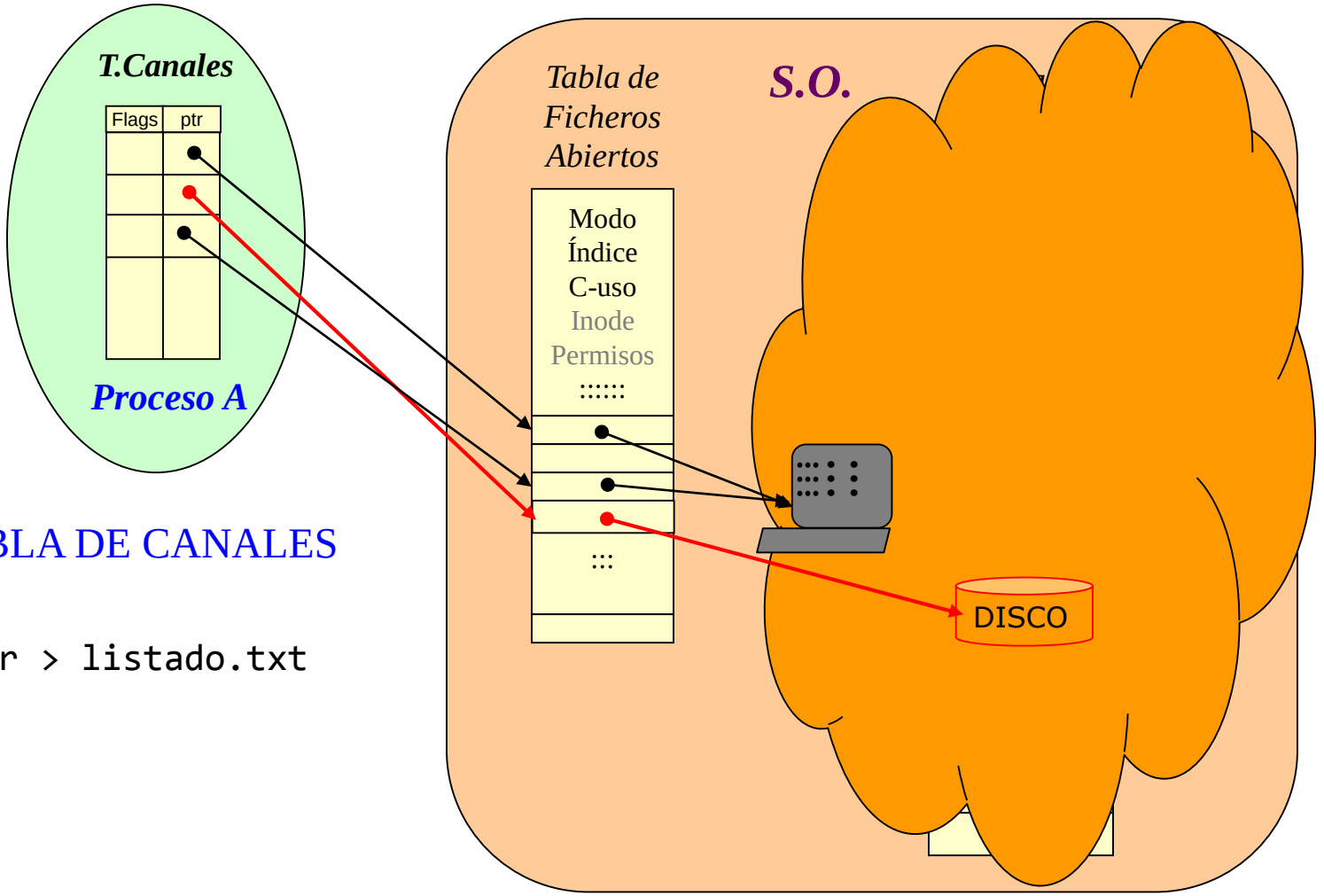


TABLA DE CANALES

```
ls dir > listado.txt
```

Independencia del Dispositivo en UNIX

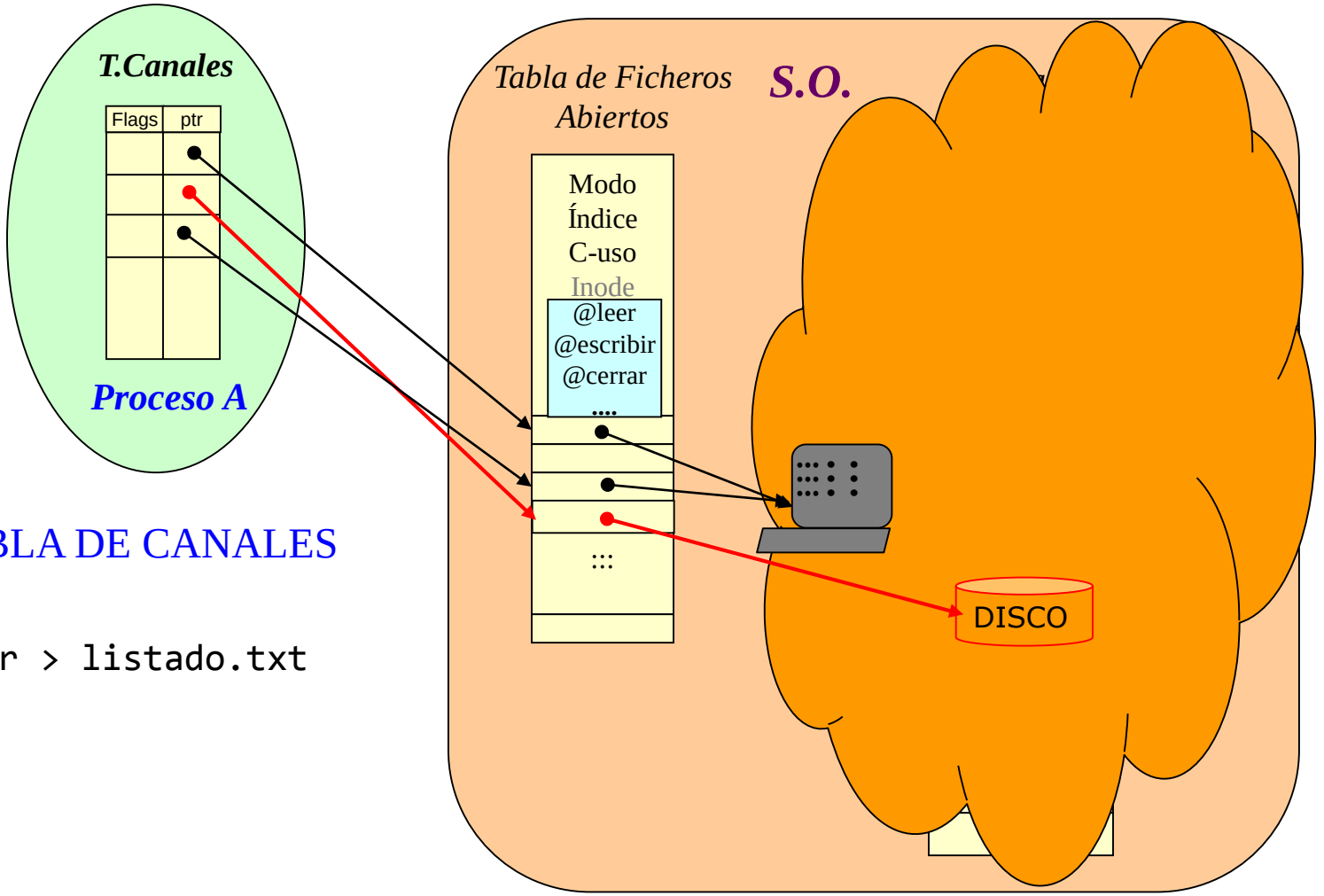
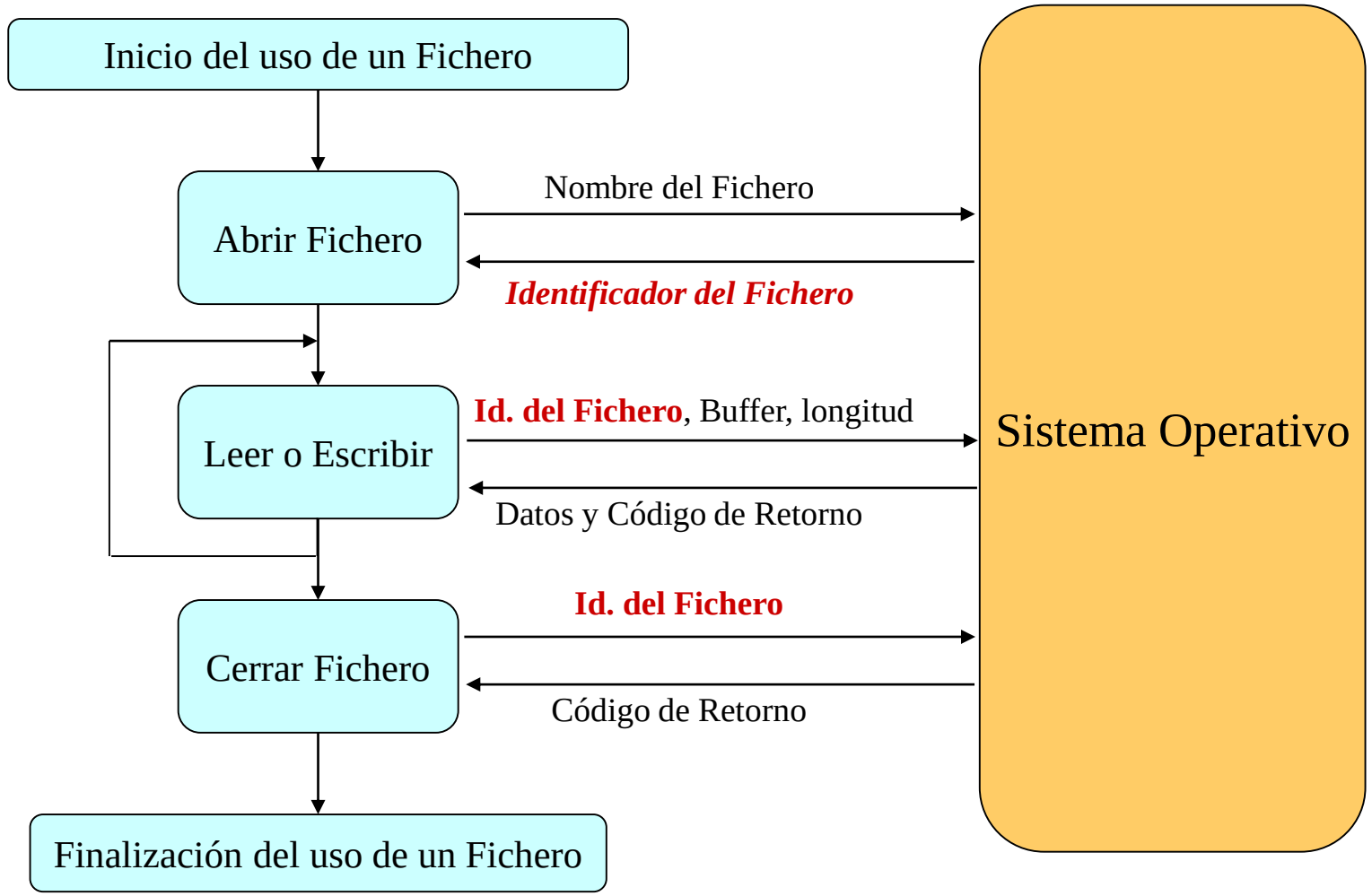


TABLA DE CANALES

```
ls dir > listado.txt
```

Pasos para usar un Fichero



Niveles de traducción de direcciones

Leer/Escribir
Registro, canal

↓

Leer/Escribir (Llamada al sistema)
byte, canal

USUARIO

Rutina Dispersora + Tabla de canales
Relativo a byte -> Bloque Relativo

↓

Leer/Escribir
Bloque, fichero o dispositivo

Información de ubicación (I-node)
Bloque relativo -> Bloque absoluto

SISTEMA

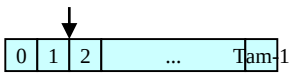
↓

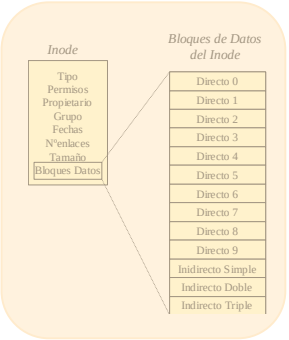
Leer/Escribir
Bloque Absoluto, dispositivo

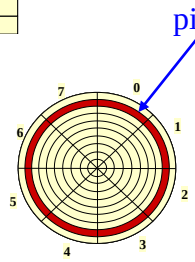
Características físicas del dispositivo
Bloque absoluto -> Cilindro, pista, sector

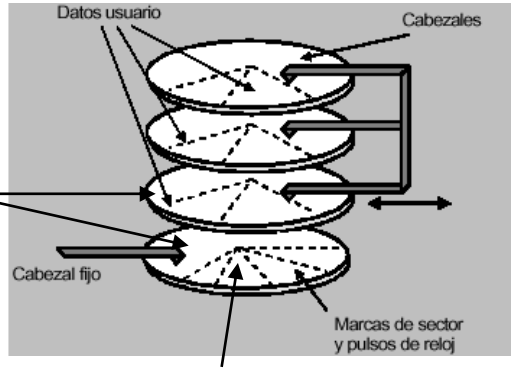
↓

Leer/Escribir
Datos registros controlador









Llamadas al Sistema para lectura/escritura de ficheros (UNIX/Linux)

- **Apertura/lectura/escritura/ubicación**

```
int open (char *camino, int flags, int perm);
int open (char *camino, int flags);
int creat (char *camino, int perm);
int close (int fd);
int read (int fd, void *buf, unsigned lon);
int write ( int fd, void *buf, unsigned lon);
long lseek (int fd, long despl, int cod_pos);
//     origen:0,     pos_actual:1,        final:2
//     SEEK_SET,     SEEK_CUR,            SEEK_END
```

Flags:

O_RDONLY	**O_WRONLY**	**O_RDWR**	**O_APPEND**
O_CREAT	O_NDELAY	O_DSYNC	O_SYNC
O_NOCTTY	O_EXCL	O_TRUNC,	…

Funciones de Biblioteca de C

- **Apertura / lectura / escritura**

```
#include <stdio.h>
FILE * fopen (char *camino, char *tipo);
int fclose (FILE *fd);
int fread (void *buf, int tam, int num, FILE *fd);
int fwrite (void *buf, int tam, int num, FILE *fd);
int getchar();
int putchar (int c);
int printf (const char *format, ...);
int scanf (const char *format, ...);
int fscanf (FILE *fd, const char *format, ...);
int fprintf (FILE *fd, const char *format, ...);
...
```

Act 04.5: Ejemplo de utilización de los canales de entrada-salida adicionales a los estándar (ctee.c y stee.c)

- *Vamos a construir dos versiones del comando tee de Linux*.

1. Una con funciones de biblioteca (***ctee***)
2. Otra con llamadas al sistema (***stee***).

Act 04.6: Primera versión del proyecto de E/S

- *Vamos a construir la primera versión del proyecto de E/S a partir de las utilidades ctee y stee ya desarrolladas*.

Para su desarrollo, sigue las instrucciones indicadas en la documentación de la actividad 4.6