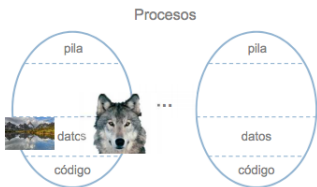


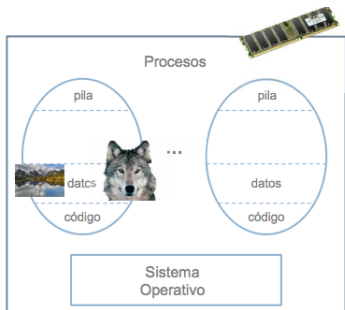
Sistemas Operativos

Sistema de archivos

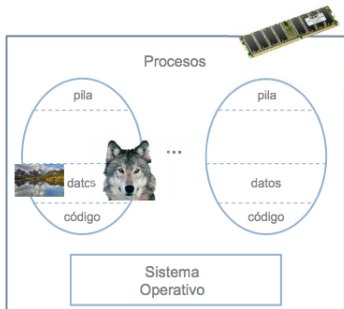
Departamento de Ingeniería en Sistemas y Computación
Universidad Católica del Norte, Antofagasta.



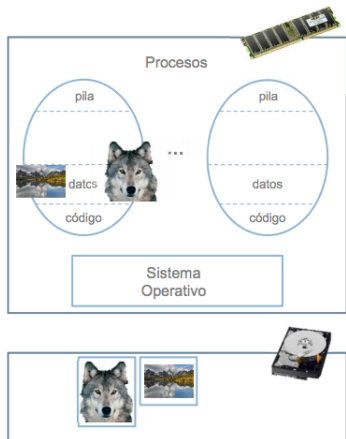
- Un proceso de edición de fotografías (por ejemplo) tiene en memoria su código y datos.
 - ▣ Cada proceso trabaja con sus datos, pudiendo generar nuevos datos.



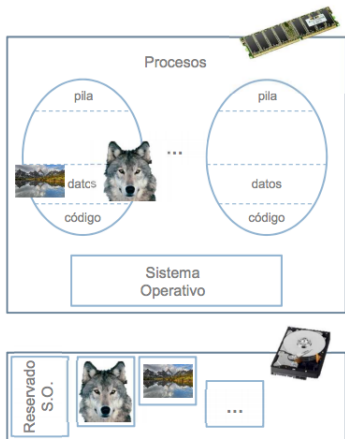
- Un proceso de edición de fotografías (por ejemplo) tiene en memoria su código y datos.
 - Cada proceso trabaja con sus datos, pudiendo generar nuevos datos.
- Puede haber varios procesos en memoria, siendo el sistema operativo el que reparte y organiza la memoria.



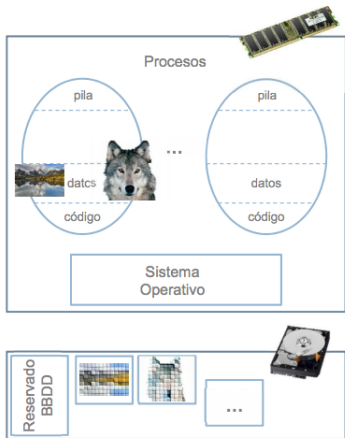
- La memoria principal en los sistemas actuales es de pequeño tamaño, acceso a palabra y volátil.
 - Los datos almacenados no son persistentes (sin electricidad).
 - Solo se usa para guardar los datos accedidos por el procesador durante un periodo.
 - Se puede acceder a cualquier palabra directamente.
- ¿Dónde guardar los datos?



- La memoria secundaria es de mayor tamaño, acceso a bloque y no volátil.
 - ▣ Datos persistentes
 - Al proceso que lo usa, a la lectura concurrente entre procesos.
 - ▣ Permitirá guardar mayor cantidad de datos que en M.P.
 - ▣ Organizada en bloques, lo que supone tener que gestionar el uso de estos bloques.
- Los datos se guardarán en M.S.: disco duro, flash, etc..

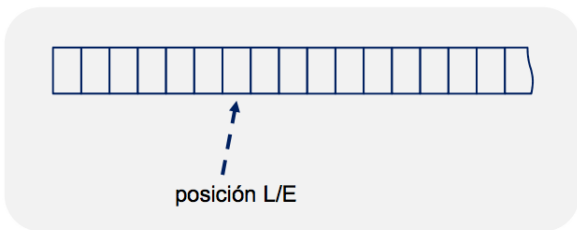


- Parte del sistema operativo se encarga de repartir y organizar la M.S.
 - ▣ Sistema de ficheros.
- El sistema de ficheros ofrece servicios para almacenar y recuperar los datos de forma simple
 - ▣ Oculta los detalles de la organización de la M.S. mediante abstracciones: ficheros, directorios, etc.

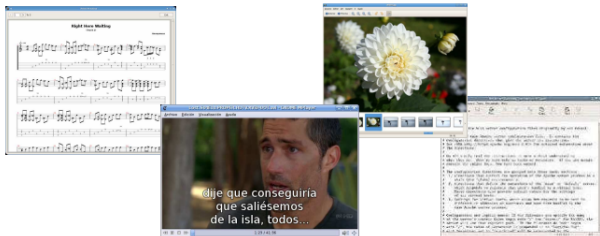


- Pero también es posible que ciertas aplicaciones organicen la M.S.:
 - ▣ Gestores de bases de datos.
- El sistema operativo ofrece acceso a todo el dispositivo.
- Es posible también una organización mixta
 - ▣ parte el sistema operativo y parte la aplicación

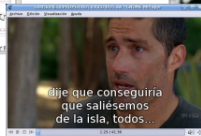
- Conjunto de información relacionada que ha sido definida por su creador.
- Habitualmente el contenido es representado por una secuencia o tira de bytes:



□ Diferentes tipos de información:



□ Diferentes tipos de estructura de esa información:



- Complejos
 - Formato (XML, etc.)
 - Reubicables



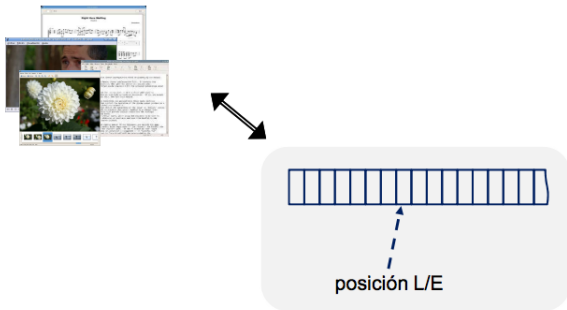
- Registros

- Longitud fija
- Longitud variable



- Secuencia de palabras

- Las aplicaciones convierten y almacenan como una **secuencia o tira de bytes**:



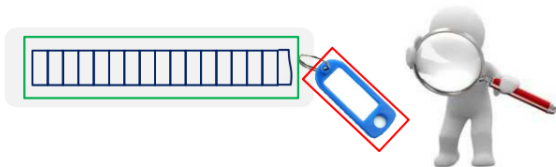
□ Información de un archivo:

□ Datos

- Información que almacena el archivo.



□ Metadatos

- Información sobre el archivo: distintos **atributos** sobre el archivo.

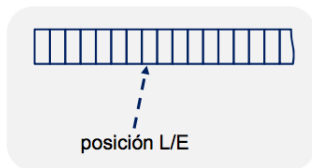


- **Atributos típicos de un fichero:**
 - **Nombre:** identificador para los usuarios del fichero.
 - **Tipo:** tipo de archivo (para los sistemas que lo necesiten)
 - Ej.: extensión (.exe, .pdf, etc.)
 - **Localización:** identificador que ayuda a la localización de los bloques del dispositivo que pertenecen al archivo.
 - **Tamaño:** tamaño actual del fichero.
 - **Protección:** control de qué usuario puede leer, escribir, etc.
 - **Día y hora:** instante de tiempo de último acceso, de creación, etc. que permite la monitorización del uso del archivo.
 - **Identificación de usuario:** identificador del creador, dueño del archivo, etc.

- Se utiliza tiras de caracteres:
 - ▣ Permite a los usuarios organizarse mejor
 - ▣ Los usuarios no recuerdan nombres del tipo 00112233

 - Es característico de cada sistema de ficheros:
 - ▣ Longitud del nombre: fijo (MS-DOS) o variable (UNIX)
 - ▣ Sensibles a mayúsculas/minúsculas (Unix) o no (MS-DOS)
 - INMA e inma
 - ▣ Necesario extensión: si y fija (MS-DOS), no (UNIX)
-  remain.zip ■ .zip -> identifica el tipo de fichero (y la aplicación a usar)
-  /bin/lis ■ file nombre -> identifica por contenido (número mágico)

- **Interfaz genérica** para acceder a la información:
 - `descriptor` ← `open` (nombre, flags, modo)
 - `close` (descriptor)
 - `read` (descriptor, puntero, tamaño)
 - `write` (descriptor, puntero, tamaño)
 - `lseek` (descriptor, desplazamiento, origen)
 - `ioctl` (descriptor, operación, puntero_a_parámetros)



escritura

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>

int main ( int argc, char *argv[] )
{
    int fd1 ;
    char str1[10] ;
    int nb ;

    fd1 = open ("/tmp/txt1",
               O_CREAT|O_RDWR, S_IRWXU);
    if (-1 == fd1) {
        perror("open:");
        exit(-1);
    }

    strcpy(str1,"hola");
    nb = write (fd1,str1,strlen(str1));
    printf("bytes escritos = %d\n",nb);

    close (fd1);
    return (0) ;
}
```

lectura

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>

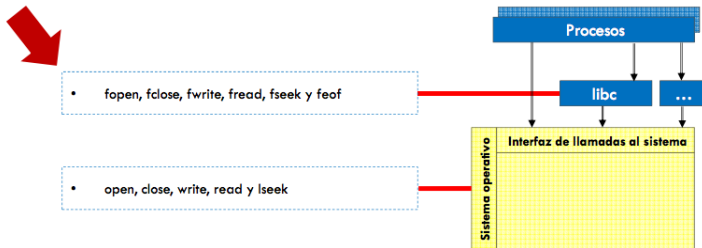
int main ( int argc, char *argv[] )
{
    int fd1 ;
    char str1[10] ;
    int nb, i ;

    fd1 = open ("/tmp/txt1",O_RDONLY);
    if (-1 == fd1) {
        perror("open:");
        exit(-1);
    }

    i=0;
    do {
        nb = read (fd1,&(str1[i]),1); i++;
    } while (nb != 0) ;
    str1[i] = '\0';
    printf("%s\n",str1);

    close (fd1);
    return (0);
}
```


Interfaz C99



escritura

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

int main ( int argc, char *argv[] )
{
    FILE *fd1 ;
    char str1[10] ;
    int nb ;

    fd1 = fopen ( "/tmp/txt2", "w+" );
    if ( NULL == fd1 ) {
        printf("fopen: error\n");
        exit(-1) ;
    }

    strcpy(str1, "mundo");
    nb = fwrite (str1, strlen(str1), 1, fd1);
    printf("items escritos = %d\n", nb);

    fclose (fd1) ;
    return (0) ;
}
```

lectura

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

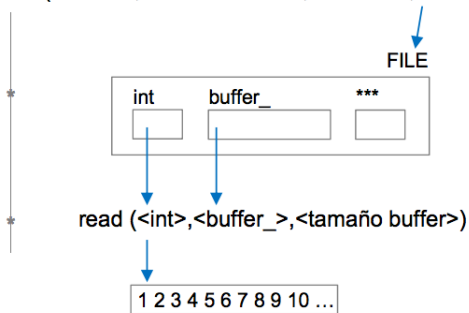
int main ( int argc, char *argv[] )
{
    FILE *fd1 ;
    char str1[10] ;
    int nb, i ;

    fd1 = fopen ( "/tmp/txt2", "r" );
    if ( NULL == fd1 ) {
        printf("fopen: error\n");
        exit(-1) ;
    }

    i=0;
    do {
        nb = fread (&(str1[i]), 1, 1, fd1) ;
        i++ ;
    } while (nb != 0) ; /* feof() */
    str1[i] = '\0' ;
    printf("%s\n", str1);

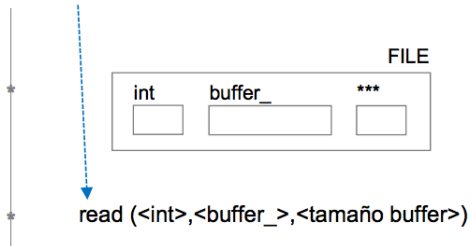
    fclose (fd1);
    return (0);
}
```

`fread (<buffer>,<tamaño 1 elto>,<nº eltos>,<FILE *>)`



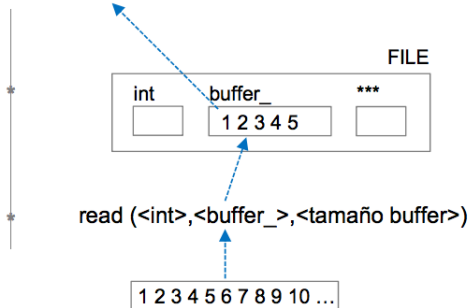
Un puntero a FILE contiene el descriptor de fichero y un buffer intermedio (principalmente)...

`fread (<buffer>,<tamaño 1 elto>,<nº eltos>,<FILE *>)`



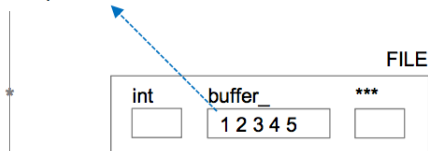
... de manera que cuando se pide la primera lectura, se realiza una lectura sobre el buffer (cuyo tamaño es mayor que el elemento pedido)...

`fread (<buffer>, <tamaño 1 elto>, <nº eltos>, <FILE *>)`



... los datos se cargan en el buffer y se copian la porción pedida al proceso que hace el fread...

`fread (<buffer>, <tamaño 1 elto>, <nº eltos>, <FILE *>)`



`read (<int>, <buffer_>, <tamaño buffer>)`

1 2 3 4 5 6 7 8 9 10 ...

...y la siguiente vez que se hace una lectura, si está en el buffer (memoria) se copia directamente de él. De esta forma se reduce las llamadas al sistema, lo que acelera la ejecución.

escritura

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>

#define BSIZE 1024

int main ( int argc, char *argv[] )
{
    FILE *fd1 ; int i; double tiempo ;
    char buffer1[BSIZE] ;
    struct timeval ti, tf;

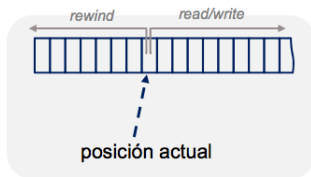
    gettimeofday (&ti, NULL);
    fd1 = fopen ("/tmp/txt2", "w+");
    if (NULL == fd1) {
        printf("fopen: error\n");
        exit(-1) ;
    }
    setbuffer(fd1,buffer1,BSIZE) ;
    for (i=0; i<8*1024; i++)
        fprintf(fd1,"%d",i);
    fclose (fd1) ;

    gettimeofday (&tf, NULL);
    tiempo= (tf.tv_sec - ti.tv_sec)*1000 +
            (tf.tv_usec - ti.tv_usec)/1000.0;
    printf("%g milisegundos\n", tiempo);
    return (0) ;
}
```

- Compilar (gcc -o b b.c) y ejecutar con
 - BSIZE=1024
 - BSIZE=0

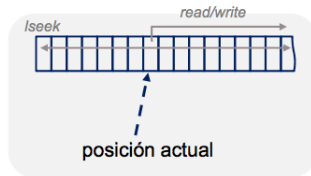
□ Acceso secuencial:

- Dispositivos de acceso secuencial: cintas magnéticas.
- Solo es posible posicionarse (*rewind*) al principio del fichero.



□ Acceso directo:

- Dispositivos de acceso aleatorio: discos duros.
- Es posible posicionarse (*lseek*) en cualquier posición del fichero.
 - Permite construir sobre él otros métodos de acceso (ej.: indexado)

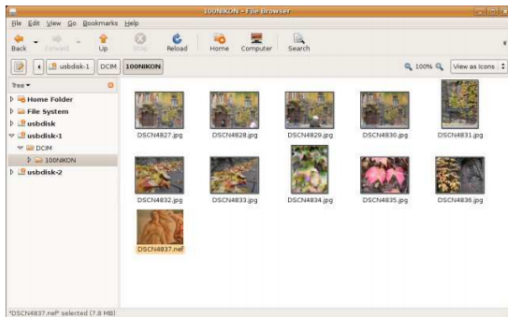
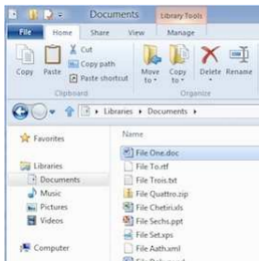


- Varios procesos pueden acceder simultáneamente a un fichero.
- Es necesario definir una semántica de coherencia:
 - ▣ ¿Cuándo son observables por otros procesos las modificaciones a un fichero?
- Opciones:
 - ▣ Semántica **UNIX**.
 - ▣ Semántica de **sesión**.
 - ▣ Semántica de **versiones**.
 - ▣ Semántica de archivos **inmutables**.

Semántica de compartición

Semántica Unix	Semántica de sesión	Semántica de versiones	Semántica inmutable
Las escrituras en un archivo son visibles inmediatamente a todos los procesos (y el nuevo puntero de L/E)	Las escrituras en un archivo no son visibles por otros procesos: al cerrar se hace visible .	Las escrituras se hacen sobre copias con número de versión: son visibles al consolidar versiones .	Si se declara compartido un archivo, no se puede modificar
Una vez abierto (<i>open</i>), la familia de procesos creado (<i>fork</i>) comparte su imagen .	Una vez cerrado el fichero, los siguientes procesos que lo abran ven las modificaciones .	Usar sincronización explícita para actualizaciones inmediatas.	Hasta no liberar el cerrojo , ni nombre ni contenido pueden modificarse.
Contención por acceso exclusivo a la imagen única del fichero.	Un fichero puede estar asociado a varias imágenes .	Tendrá varias imágenes y coste de consolidar .	No hay concurrencia .
Ext3, ufs, etc.	AFS (<i>Andrew File System</i>)	CODA	

- Estructura de datos que permite agrupar un conjunto de ficheros según el criterio del usuario.

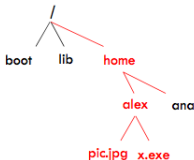


- Organizan y proporcionan información sobre la estructuración de los sistemas de archivos:



▶ De un nivel

- ▶ 1 dir con **n** ficheros
- ▶ 1 fichero con 1 dir.



▶ Jerárquico (árbol)

- ▶ 1 dir con **n** entradas
- ▶ 1 entrada con 1 dir.



▶ Árbol a-cíclico

- ▶ 1 dir. con **n** entradas
- ▶ 1 entrada con **n** dir.

□ Nombres jerárquicos para la identificación.

□ Tipo de nombrado de directorio:

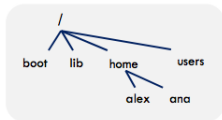
□ **Nombre absoluto:** especificación del nombre respecto al directorio raíz (/ en LINUX, \ en Windows)

□ **Nombre relativo:** especificación del nombre respecto a un directorio distinto del raíz.

■ Ejemplo: (estando en /users/) alex/correo.txt

■ Relativos al directorio de trabajo o actual:

basado en el directorio en el que se encuentre el usuario (directorio de trabajo)



□ Directorios especiales:

□ Directorio actual o directorio de trabajo: . (Ej.: cp /alex/correo.txt .)

□ Directorio padre: .. (Ej.: ls ..)

□ Directorio base del usuario: **\$HOME** (Ej.: ls -las \$HOME)

□ Información de un directorio:

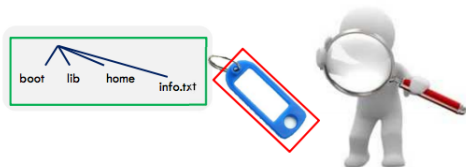
□ Datos

fichero | directorio

- "fichero especial" cuyo contenido es un listado con los **entradas** que contiene.

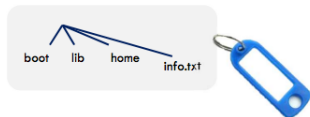
□ Metadatos

- Información sobre el directorio en sí: distintos **atributos** sobre el directorio.



□ Atributos típicos de un directorio:

- **Nombre:** identificador para los usuarios del directorio.
- **Tamaño:** número de ficheros en el directorio.
- **Protección:** control de qué usuario puede leer, acceder, etc.
- **Día y hora:** instante de tiempo de último acceso, de creación, etc. que permite la monitorización del uso del directorio.
- **Identificación de usuario:** identificador del creador, etc.

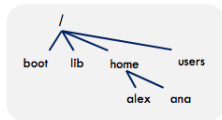


□ Interfaz genérica para gestión de directorios:

- **mkdir** (nombre,modo)
- **rmdir** (nombre)
- **chdir** (nombre)
- **getcwd** (nombre, tamaño_nombre)

- descriptor ← **opendir** (nombre)
- **closedir** (descriptor)
- estructura ← **readdir** (descriptor)
- **rewindir** (descriptor)

- **unlink** (nombre)
- **rename** (antiguo_nombre, nuevo_nombre)



lectura de /tmp

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

int main ( int argc, char *argv[] )
{
    DIR *dir1 ;
    struct dirent *dp ;
    char nombre[256] ;
    int ret ;

    ret = chdir ("/tmp/") ;
    if (ret < 0) exit(-1) ;

    getcwd (nombre, 256) ;
    printf("%s\n", nombre) ;

    dir1 = opendir (nombre) ;
    if (NULL == dir1) exit(-1) ;
    while ( (dp = readdir (dir1)) != NULL) {
        printf("%s\n", nombre, dp->d_name) ;
    }
    closedir (dir1) ;

    return (0) ;
}
```

Cambiar de directorio de trabajo

Imprimir el directorio actual de trabajo

Abrir un directorio para trabajar con él

Leer entradas del directorio e imprimir el nombre de cada entrada

Cerrar el directorio de trabajo

lectura de argv[1]

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <stdio.h>

int main ( int argc, char *argv[] )
{
    DIR *dir1 ;
    struct dirent *dp ;
    struct stat s ;

    dir1 = opendir (argv[1]);
    if (NULL == dir1) {
        perror("opendir:");
        return (-1);
    }

    while ( (dp = readdir (dir1)) != NULL) {
        stat(dp->d_name, &s);
        if (S_ISDIR(s.st_mode))
            printf("dir: %s\n", dp->d_name);
        else printf("fch: %s\n", dp->d_name);
    }

    closedir (dir1);
    return (0) ;
}
```

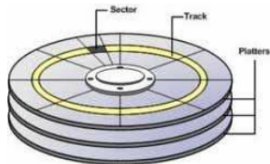
Abrir un directorio para trabajar con él

Leer entradas del directorio...

...para cada entrada obtener los metadatos de la misma e imprimir si es fichero o directorio junto con el nombre de la entrada

Cerrar el directorio de trabajo

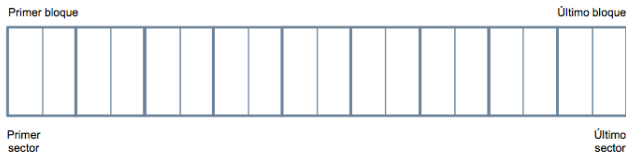
- El dispositivo de almacenamiento se divide en **sectores**, pistas y cilindros.



Primer sector

Último sector

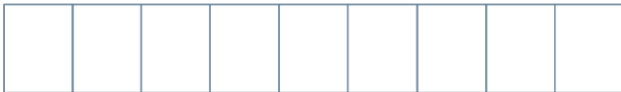
- **Bloque:** agrupación lógica de sectores de disco (2^n sectores)
 - ▣ Es la unidad de transferencia mínima usado por el S.O.
 - ▣ Optimizar la eficiencia de la entrada/salida de los dispositivos.
 - ▣ Los usuarios pueden definir el tamaño de bloque al crear el sistema de ficheros, o usar el ofrecido por defecto en el S.O.



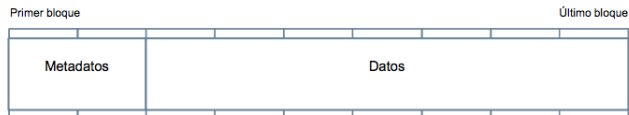
- **Bloque:** *agrupación lógica de sectores de disco (2ⁿ sectores)*
 - Es la unidad de transferencia mínima usado por el S.O.
 - Optimizar la eficiencia de la entrada/salida de los dispositivos.
 - Los usuarios pueden definir el tamaño de bloque al crear el sistema de ficheros, o usar el ofrecido por defecto en el S.O.

Primer bloque

Último bloque



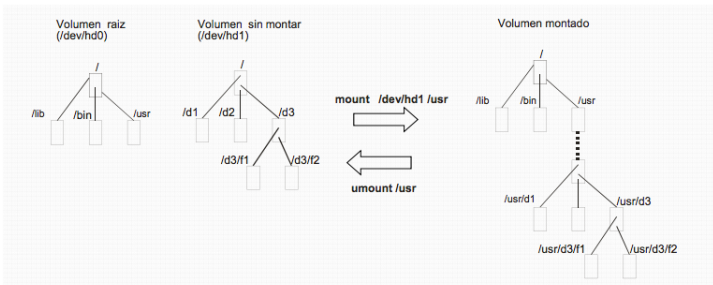
- El sistema de archivos permite organizar la información dentro de los dispositivos de almacenamiento en un formato inteligible para el sistema operativo:
 - ▣ Es un *conjunto coherente de metainformación y datos*.



- **Atributos típicos de un sistema de fichero:**
 - **Tamaños usados:**
 - **Número de bloques:** cantidad de bloques gestionados (datos + metadatos)
 - **Tamaño de bloque:** tamaño del bloque (en bytes o en sectores).
 - **Número de entradas:** número de entradas (ficheros y directorios) gestionados.
 - **Tamaño de la zona de metadatos:** número de bloques dedicados.
 - **Gestión de espacio libre:** identificación de qué bloque está libre.
 - **Gestión de entradas:** para cada entrada (fichero o directorio) se reserva un espacio para los metadatos que la describe:
 - **Atributos generales:** fechas, permisos, identificación de usuario, etc.
 - **Atributos para la gestión de ocupado:** bloques usados por esta entrada.
 - **Referencia a la entrada del directorio raíz:** identificación de la entrada que contiene la información del directorio raíz.

□ Operaciones con sistemas de ficheros:

- Crear
- Montar
- Desmontar



□ Gran cantidad de sistemas de ficheros.

□ Para dispositivos de almacenamiento:

- minix (Minix)
- ext2 (Linux)
- ext3 (Linux)
- ufs (BSD)
- fat (DOS)
- vfat (win 95)
- hpfs (OS/2)
- hfs (Mac OS)
- ntfs (win NT/2K/XP)
- ...

□ Especiales:

- procs (/proc)
- devFS (/dev)
- umsdos (Unix sobre DOS)
- ...

□ En red:

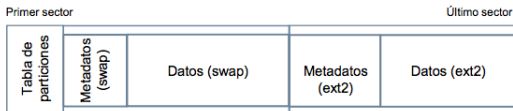
- NFS
- CODA
- SMBFS
- NCPFS (Novell)
- ...

- ▶ Contenedor de un sistema de ficheros.



- ▶ Una **partición** es una porción de un disco a la que se la dota de una identidad propia y que puede ser manipulada por el sistema operativo como una entidad lógica independiente.

- ▶ Contenedor de un sistema de ficheros.



- ▶ Una vez creadas las particiones, el sistema operativo debe crear las estructuras de los sistemas de archivos dentro de esas particiones.

Para ello se proporcionan mandatos como `format` o `mkfs` al usuario:

- ▶ `# mkswap -c /dev/hda1 20800`
- ▶ `# mkfs -c /dev/hda2 -b 8196 123100`

- **Atributos típicos de una partición:**
 - **Tipo:** primaria, secundaria, unidad lógica, con arranque, etc..
 - **Tamaño:** inicio y fin de partición.
 - **Sistema albergado:** linux, linux swap, vfat, etc.
 - **Identificación:** número de partición (orden o UUID).



- Disco duro
- SSD (estado sólido)
- Sistemas ópticos



❑ Listar los dispositivos PCI:

```
acaldero@phoenix:~/infodso/$ lspci
```

```
00:00.0 Host bridge: Intel Corporation 82Q35 Express DRAM Controller (rev 02)
00:01.0 PCI bridge: Intel Corporation 82Q35 Express PCI Express Root Port (rev 02)
00:03.0 Communication controller: Intel Corporation 82Q35 Express MEI Controller (rev 02)
00:03.2 IDE interface: Intel Corporation 82Q35 Express PT IDER Controller (rev 02)
00:03.3 Serial controller: Intel Corporation 82Q35 Express Serial KT Controller (rev 02)
...
```

❑ Listar los dispositivos USB:

```
acaldero@phoenix:~/infodso/$ lsusb
```

```
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
...
Bus 008 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 002: ID 1241:1166 Belkin MI-2150 Trust Mouse
Bus 005 Device 002: ID 0c45:600d Microdia TwinkleCam USB camera
```