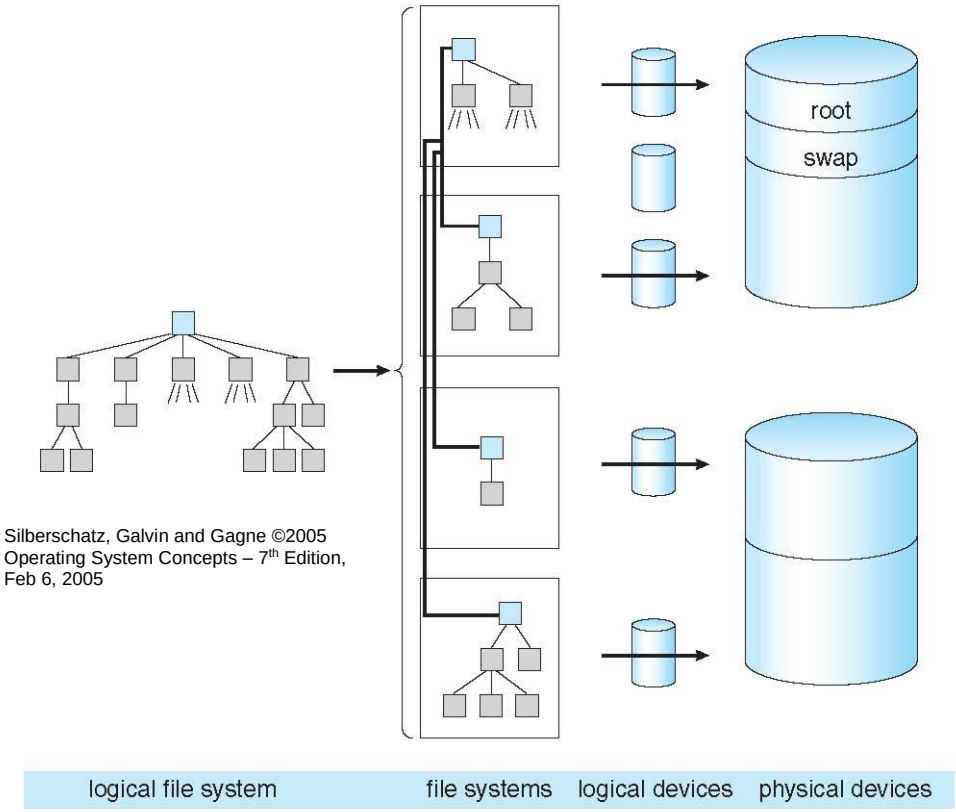
# Unix File System

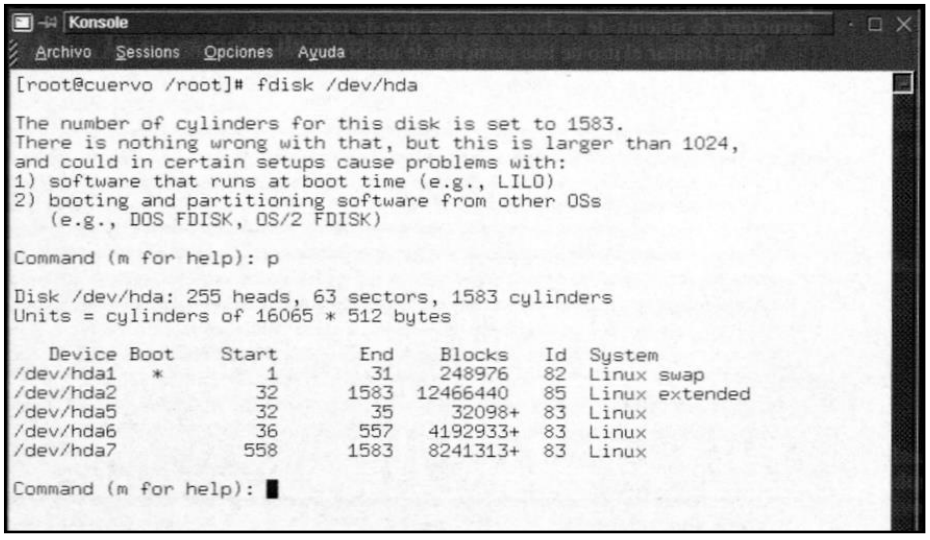
# 1. Introduction to the UNIX File System: logical vision



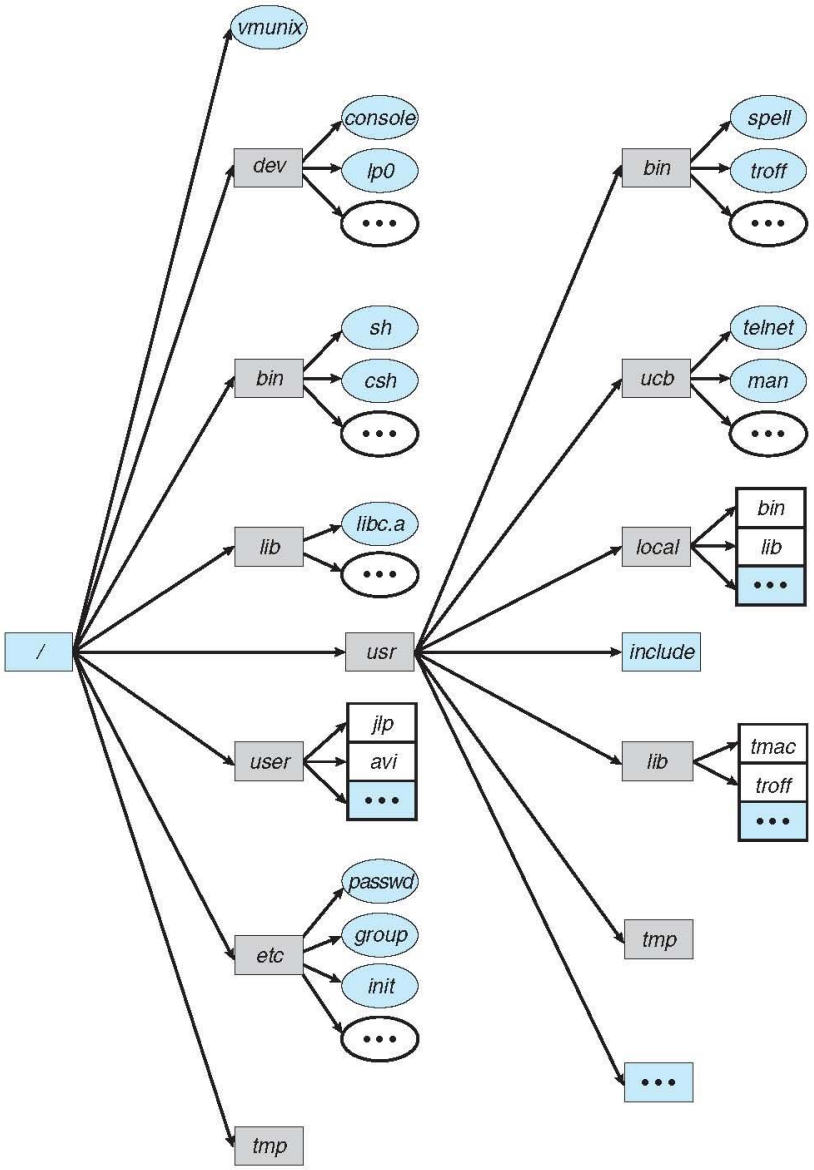

Silberschatz, Galvin and Gagne ©2005
Operating System Concepts – 7th Edition,
Feb 6, 2005

Logical structure in each FS (System V):

| BOOT | SUPERBLOCK | INODE LIST | DATA AREA |
|---|---|---|---|

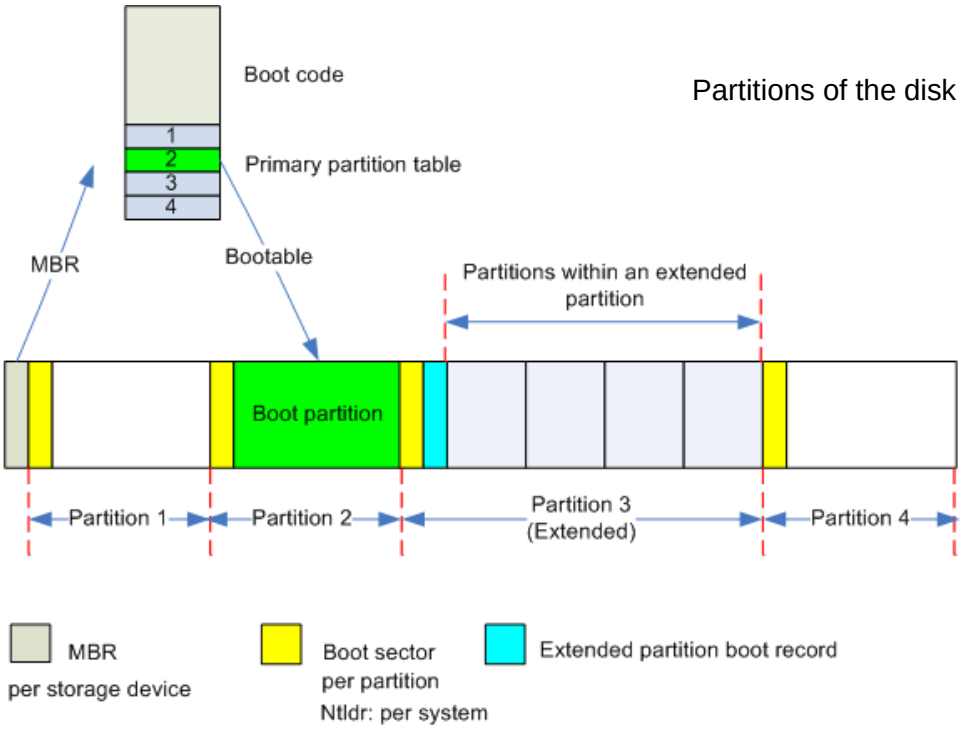



Related commands: *du*, *df, mount, umount, mkfs*




Typical directory structure in an UNIX platform.

# 2. Introduction to the UNIX File System: physical vision of disk partitions




Partitions of the disk in a PC

| Offset | Tamaño | Descripción |
|---|---|---|
| 0x000 | 446 bytes | reservado |
| 0x1BE | 16 bytes | Entrada partición 1 |
| 0x1CE | 16 bytes | Entrada partición 2 |
| 0x1DE | 16 bytes | Entrada partición 3 |
| 0x1EE | 16 bytes | Entrada partición 4 |
| 0x1FE | 2 bytes | 0xAA55 |

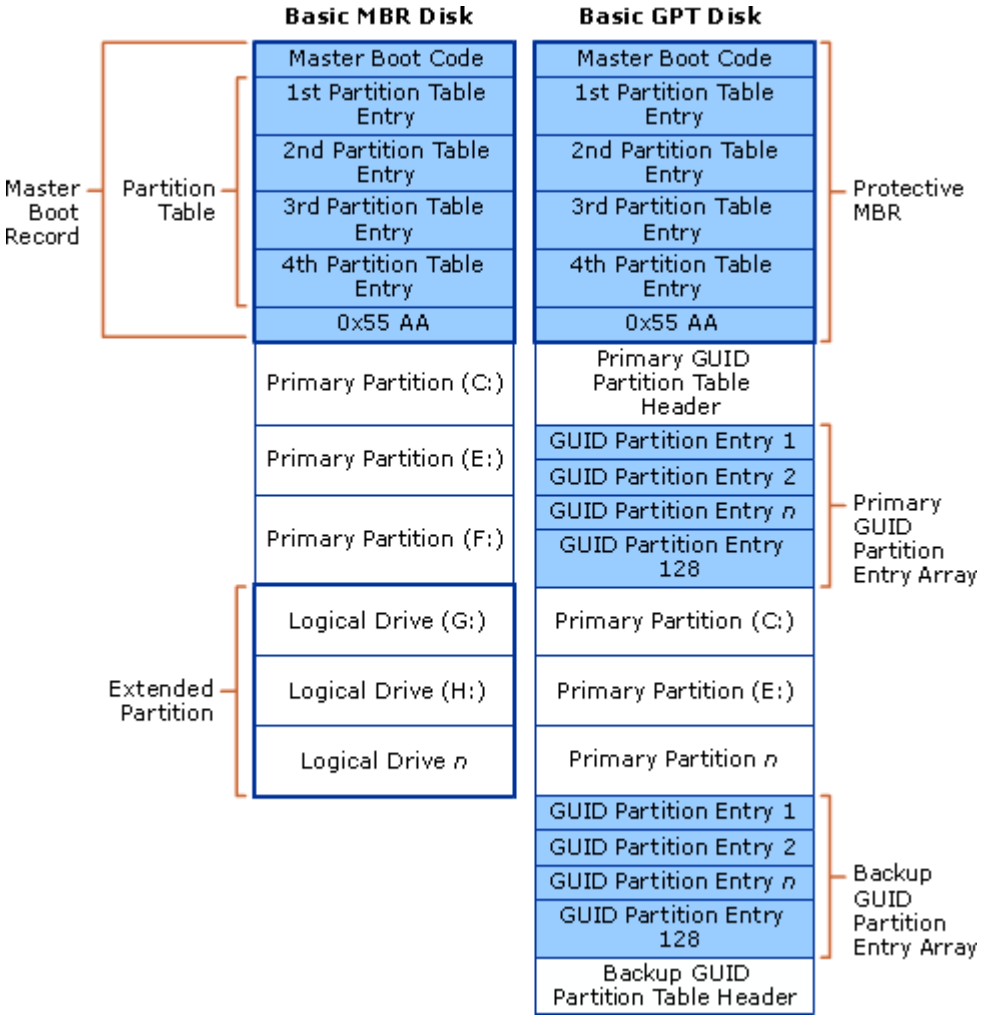
*Master Boot Record* structure

**Tipos de partición**

| Tipo de partición | Valor | Tipo de partición | Valor |
|---|---|---|---|
| Vacío | 00 | Novell Netware 386 | 65 |
| FAT de 12 bits de DOS | 01 | PIC/IX | 75 |
| XENIX (root) | 02 | MINIX antigua | 80 |
| XENIX (usr) | 03 | Linux/MINUX | 81 |
| DOS 16 bits <=32M | 04 | Linux (swap) | 82 |
| Extendida | 05 | Linux (nativa) | 83 |
| DOS 16 bits >=32 | 06 | Linux (extendida) | 85 |
| OS/2 HPFS | 07 | Amoeba | 93 |
| AIX | 08 | A noeba BBT | 94 |
| AIX (arranque) | 09 | BSD/386 | a5 |
| OS/2 Boot Manager | 0a | OpenBSD | a6 |
| FAT32 de Windows 95 | 0b | NEXTSTEP | a7 |
| FAT32 de Windows 95 (LBA) | 0c | BSDI fs | b7 |
| FAT1 de Windows 95 (LBA | 0e | BSDI swap | b8 |
| Win95 (Extendida, LBA) | 0f | Syrinx | c7 |
| Venix 80286 | 40 | CP/M | db |
| Novell? | 51 | DOS access | e1 |
| Microport | 52 | DOS R/O | e3 |
| GNU HURD | 63 | Secundaria del DOS | f2 |
| Novell Netware 286 | 64 | BBT | ff |

| Offset | Tamaño | Descripción |
|---|---|---|
| 0x0 | 1 byte | 0x80 partición activa; 0x0 inactiva |
| 0x1 | 1 byte | Cabeza del primer sector de la partición |
| 0x2 | 2 bytes | Cilindro primer sector partición (10 bytes)<br>Numero de sector del primer sector (6 bytes) |
| 0x4 | 1 byte | Tipo de partición<br>0x1 DOS primario con FAT12<br>0x4 DOS primario con FAT16<br>0x5 DOS extendido<br>0x6 DOS mayor 32M |
| 0x5 | 1 byte | Cabeza último sector de la partición |
| 0x6 | 2 bytes | Cilindro/sector último sector (como off 0x2) |
| 0x8 | 4 bytes | Sector inicial (relativo al comienzo disco) |
| 0xC | 4 bytes | Número de sectores de la partición |

Information in each partition




The widespread MBR partitioning scheme, dating from the early 1980s, imposed limitations which affected the use of modern hardware. Intel therefore developed a new partition-table format in the late 1990s, GPT, which most current OSs support.

## 2.1 System V vs. BSD (Fast File System)

Logical structure in each FS (System V):

| BOOT | SUPERBLOCK | INODE LIST | DATA AREA |
|---|---|---|---|

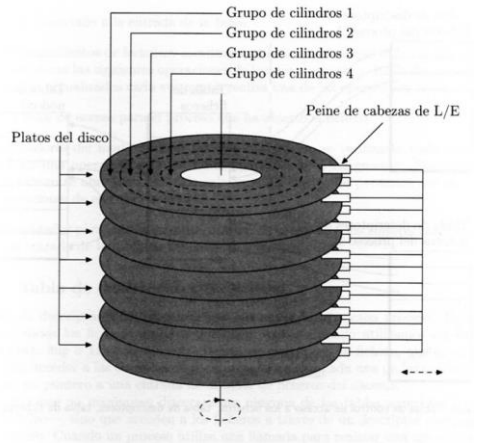
Logical structure in each FS (BSD):

| BOOT | SUPERBLOCK | CILINDER GROUP *0* | CILINDER GROUP*1* | …… | CILINDER GROUP *N* |
|---|---|---|---|---|---|

CG *i*

| SUPERBLOCK (replicated) | CILINDER GROUP *i* HEAD | INODE LIST of CILINDER GROUP *i* | DATA AREA of CILINDER GROUP *i* |
|---|---|---|---|

Organization of the disk in cylinder groups [Márquez, 2004]




BSD: Blocks and fragments. BSD uses blocks and a possible last “fragment” to assign data space for a file in the data area.

Example:

All the blocks of a file are of a large block size (such as 8K), except the last.

The last block is an appropriate multiple of a smaller fragment size (i.e., 1024) to fill out the file.

Thus, a file of size 18,000 bytes would have two 8K blocks and one 2K fragment (which would not be filled completely).

# 3. Internal representation of files

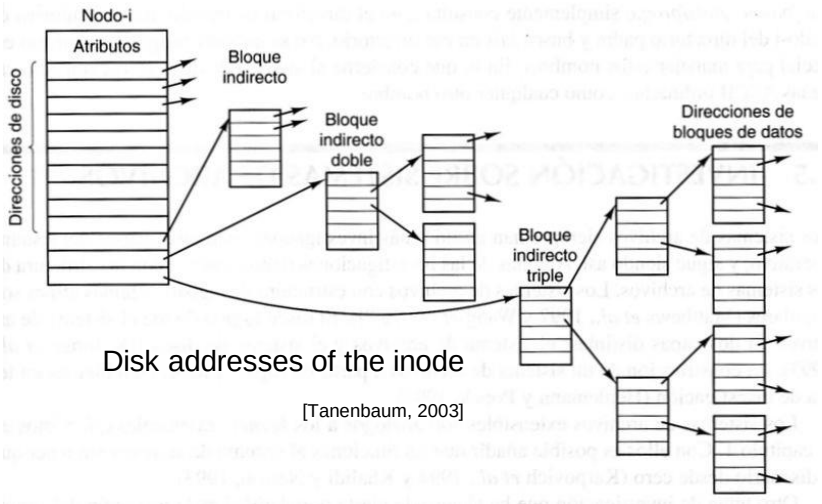
## 3.1 Inodes

- The operating system associates an inode to each file.
- We have to differentiate between:
  - Inodes in disk, in the Inode List.
  - In memory, in the Inode Table, with a similar structure to the Buffer Cache.

| Inode in disk |
|---|
| OWNER |
| GROUP |
| FILE TYPE |
| ACCESS PERMISSIONS |
| FILE DATES: access, data modification, inode modification |
| Number of LINKS |
| SIZE |
| DISK ADDRESSES |

## 3.2 Structure of the block layout in the disk

- A file has associated:
  - An inode of the Inode List.
  - Blocks of the data area. These blocks of the file are information contained in the inode file, with the following scheme:




Disk addresses of the inode

[Tanenbaum, 2003]

**Details:**

- Using blocks of 1K and addresses of 4 bytes, the maximum size is: 10K + 256K + 64M + 16G
- Slower access to larger files.

## 3.3 File types & file permissions

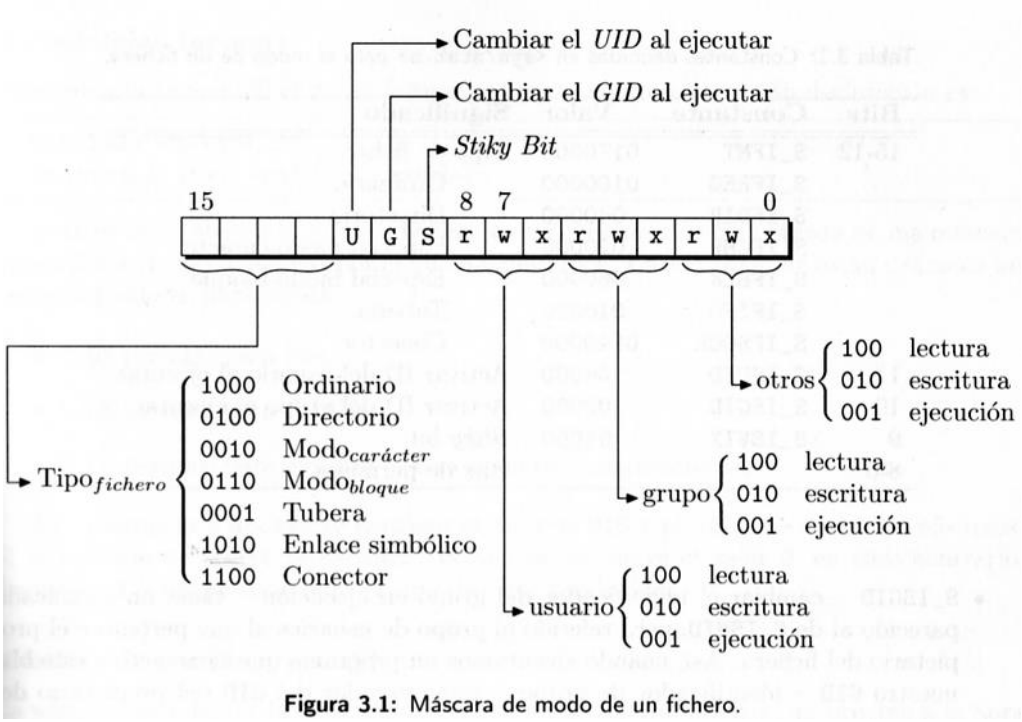



**Figura 3.1:** Máscara de modo de un fichero.

**Tabla 3.1:** Constantes definidas en <sys/stat.h> para el modo de un fichero.

| Bits | Constante | Valor | Significado |
|---|---|---|---|
| 15-12 | S_IFMT | 0170000 | Tipo de fichero: |
| | S_IFREG | 0100000 | Ordinario |
| | S_IFDIR | 040000 | Directorio |
| | S_IFCHR | 020000 | Especial modo carácter |
| | S_IFBLK | 060000 | Especial modo bloque |
| | S_IFIFO | 010000 | Tubería |
| | S_IFSOCK | 0140000 | Conector |
| 11 | S_ISUID | 04000 | Activar ID del usuario al ejecutar |
| 10 | S_ISGID | 02000 | Activar ID del grupo al ejecutar |
| 9 | S_ISVTX | 01000 | *Stiky* bit |
| 8-0 | | | Bits de permisos |

Related command (and system call) to the file mode: *chmod*
Related command (and system call) to the file owner *chown*

# 4. Directories

- A directory is a file whose content is interpreted as “directory entries”.
- Directory entry format:

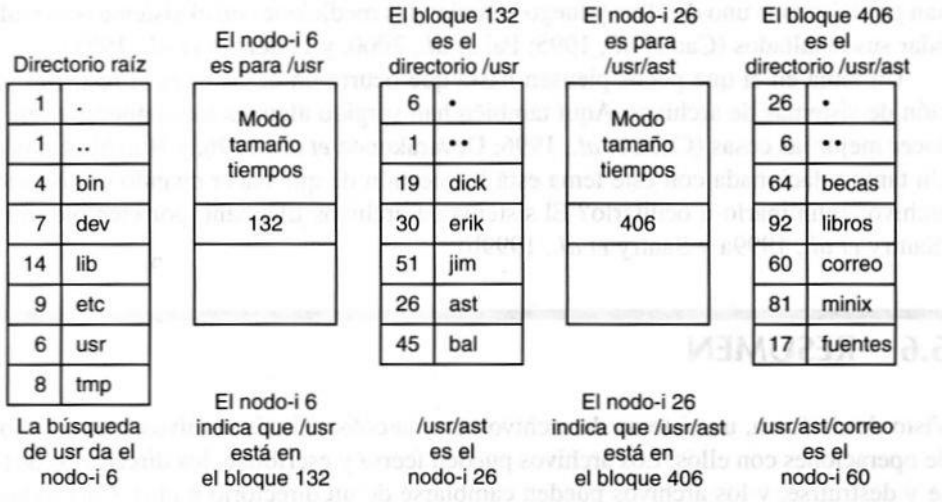
System V directory entry:

| Inode number (2 bytes) | Name (14 bytes) |
|---|---|

BSD directory entry:

| Inode number (4 bytes) | Length of the entry (2 bytes) | Length of the file name (2 bytes) | Name ( '\0'-ended until a length multiple of 4) (variable) |
|---|---|---|---|

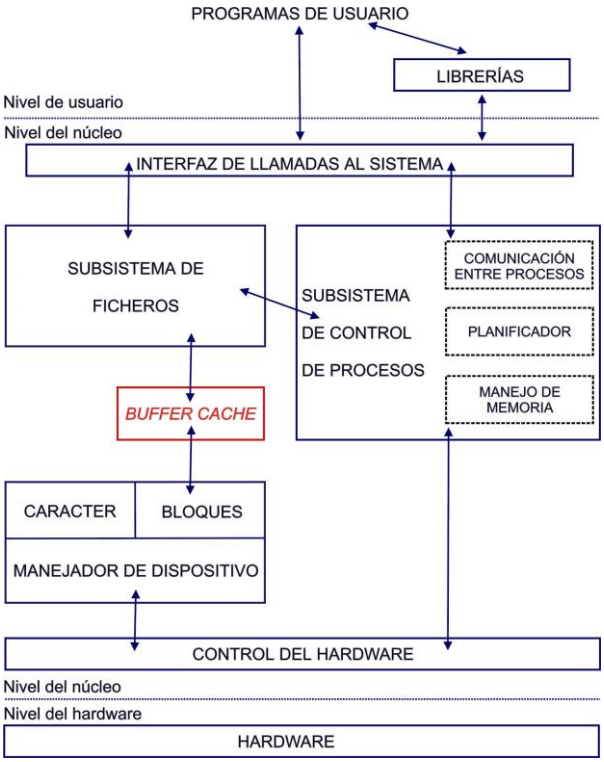
Related system calls: *opendir*, *readdir, closedir (defined in <dirent.h>)*




Example of the necessary steps in the search of the inode of the file */usr/ast/correo*

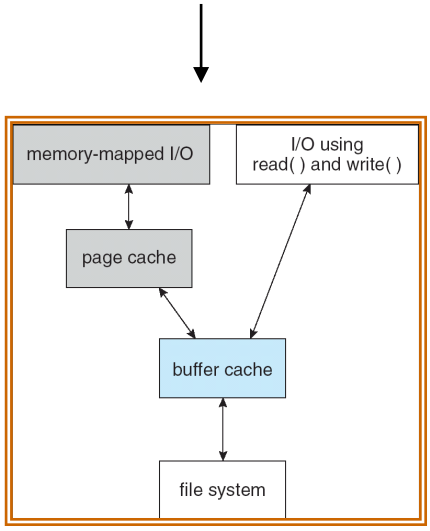
[Tanenbaum, 2003]

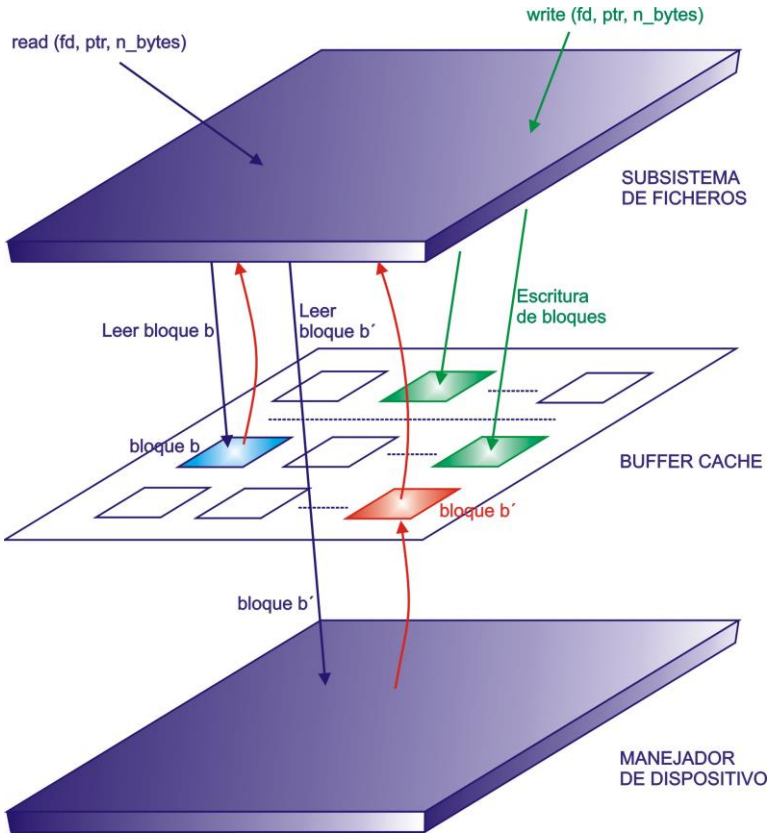
# 5. Brief description of the kernel structures related to the file system

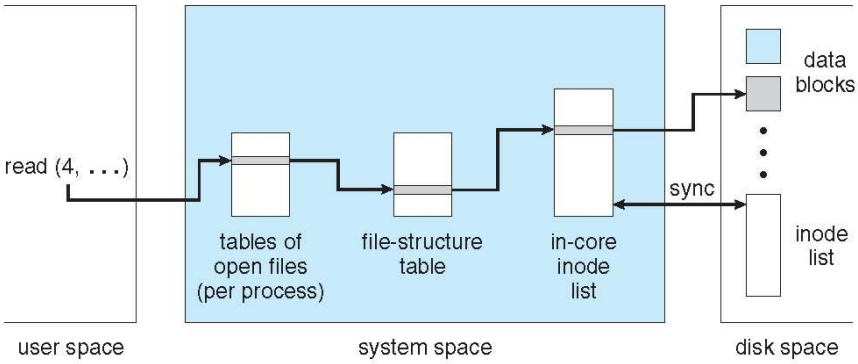



Block diagram of the system kernel.

The buffering mechanism of the *Buffer Cache* regulates data flow between secondary storage block devices and the kernel, decreasing the number of accesses to the disk. There is a similar mechanism associated to virtual memory with a *Page Cache*.






Scheme of the main kernel structures related to the file system

(Silberschatz, Galvin and Gagne ©2005 Operating System Concepts – 7th Edition, Feb 6, 2005)

**SYSTEM CALLS FOR THE FILE SYSTEM**

File System Calls

| Return File Desc | Use of namei | Assign inodes | File Attributes | File I/O | File Sys Structure | Tree Manipulation |
|---|---|---|---|---|---|---|
| open<br>creat<br>dup<br>pipe<br>close | open stat<br>creat link<br>chdir unlink<br>chroot mknod<br>chown mount<br>chmod umount | creat<br>mknod<br>link<br>unlink | chown<br>chmod<br>stat | read<br>write<br>lseek | mount<br>umount | chdir<br>chown |

| Lower Level File System Algorithms | | |
|---|---|---|
| namei<br>iget iput | ialloc ifree | alloc free bmap |
| buffer allocation algorithms | | |
| getblk brelse bread breada bwrite | | |

# 6. System calls for the file system

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

open mode:

mode 0: read
mode 1: write
mode 2: read-write

Or using the constatnts defined in the header <fcntl.h>

| | |
|---|---|
| O_RDONLY | only read |
| O_RDWR | read-write |
| O_WRONLY | only write |
| O_APPEND | append |
| O_CREAT | create |
| ... | |

*int read (int df, char *buff, int n);*

*df* – file descriptor *open* returns
*buff* – address, in the user space, where the data are transferred
*n* – number of bytes to be read

*int write (int df, char *buff, int n);*

Example of openings from two processes:

Proc A:

*fd1=open("/etc/passwd", O_RDONLY);*

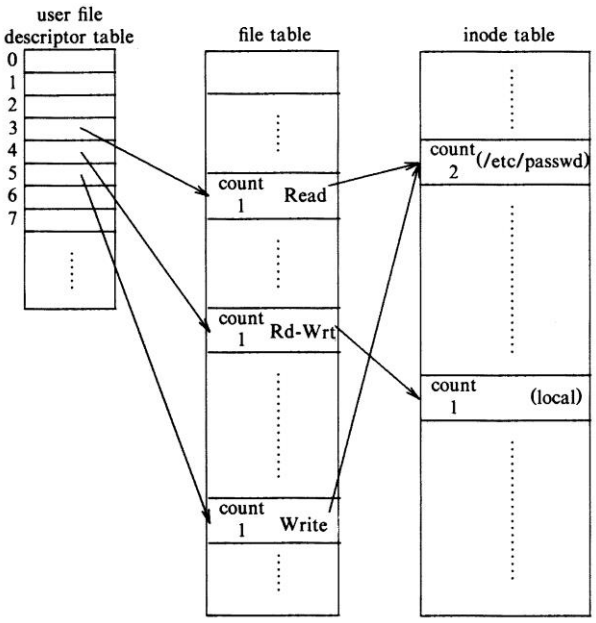
*fd2=open("local", O_RDWR);*

*fd3=open("/etc/passwd", O_WRONLY);*

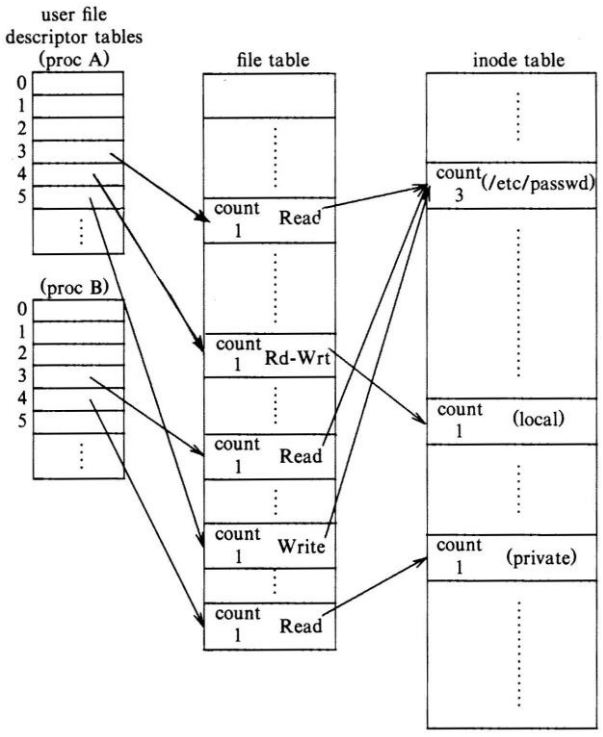
Proc B:

*fd1=open("/etc/passwd", O_RDONLY);*

*fd2=open("private", O_RDONLY);*




Data structures after the openings of Proc A




Data structures after the two processes opened the files



[Batch, 1986] Bach, M.J., The Design of the UNIX Operating System, Prentice-Hall, 1986.

*int newfd= dup (int df);*

*df* – file descriptor of an open file
*newfd* – new file descriptor that references the same file

*dup2(fd, newfd);*

Example:

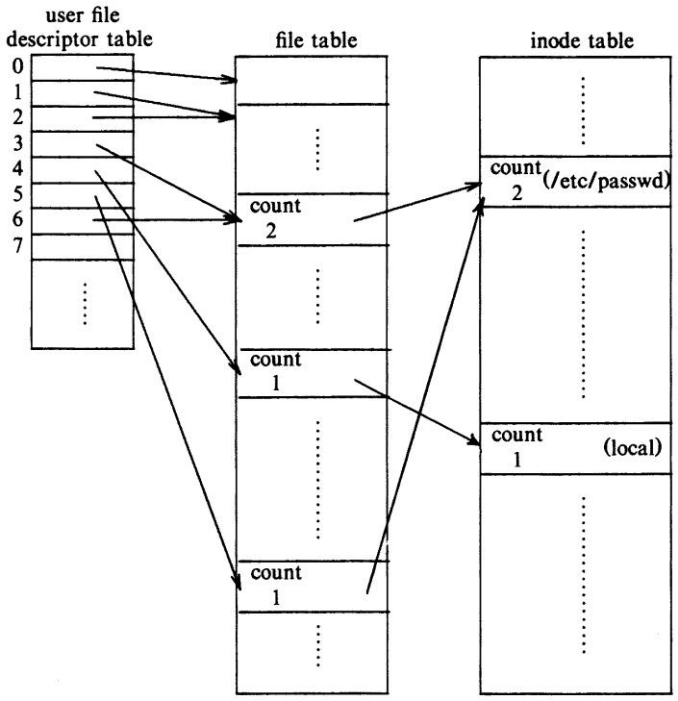
*fd1=open("/etc/passwd", O_RDONLY);*

*fd2=open("local", O_RDWR);*

*fd3=open("/etc/passwd", O_WRONLY);*

*dup(fd3);*

It returns the first free file descriptor, number 6 in this case




Data structures after dup

[Batch, 1986] Bach, M.J., The Design of the UNIX Operating System, Prentice-Hall, 1986.

# 7. SETUID executables

The kernel associates two user IDs to a UNIX process:

1. The *real user* ID: user who runs the process.
2. The *effective user* ID: used to check file access permissions, to assign ownership of newly created files and to check permission to send signals.

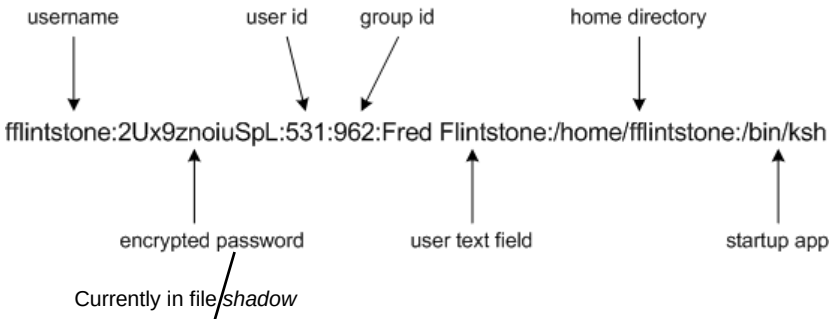
The kernel allows a process to change its effective used ID when it execs a "*setuid program*" or when it invokes the *setuid()* system call explicitly.

A SETUID program is an executable file that has the SETUID bit set in its permission model field. **When a setuid program is executed, the kernel sets the effective user ID to the owner of the executable file**.

Example of application: command *passwd*

Files in /etc:

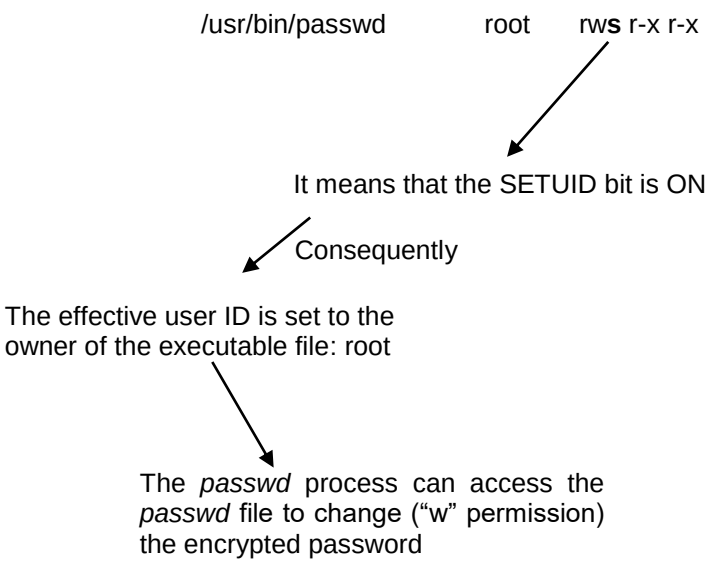
rw- r-- r-- root root *passwd* users defined in the system

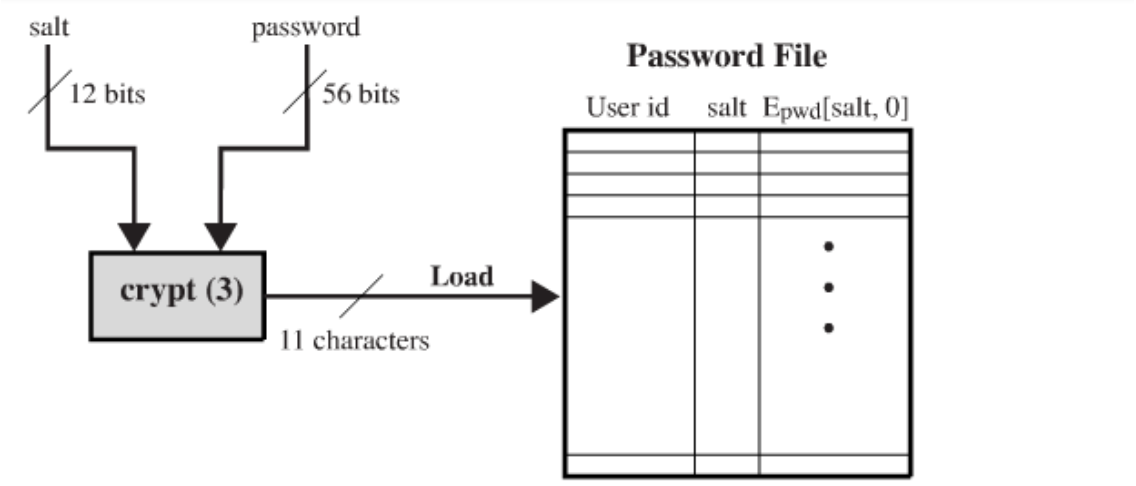



rw- r-- --- root shadow *shadow* encrypted passwords

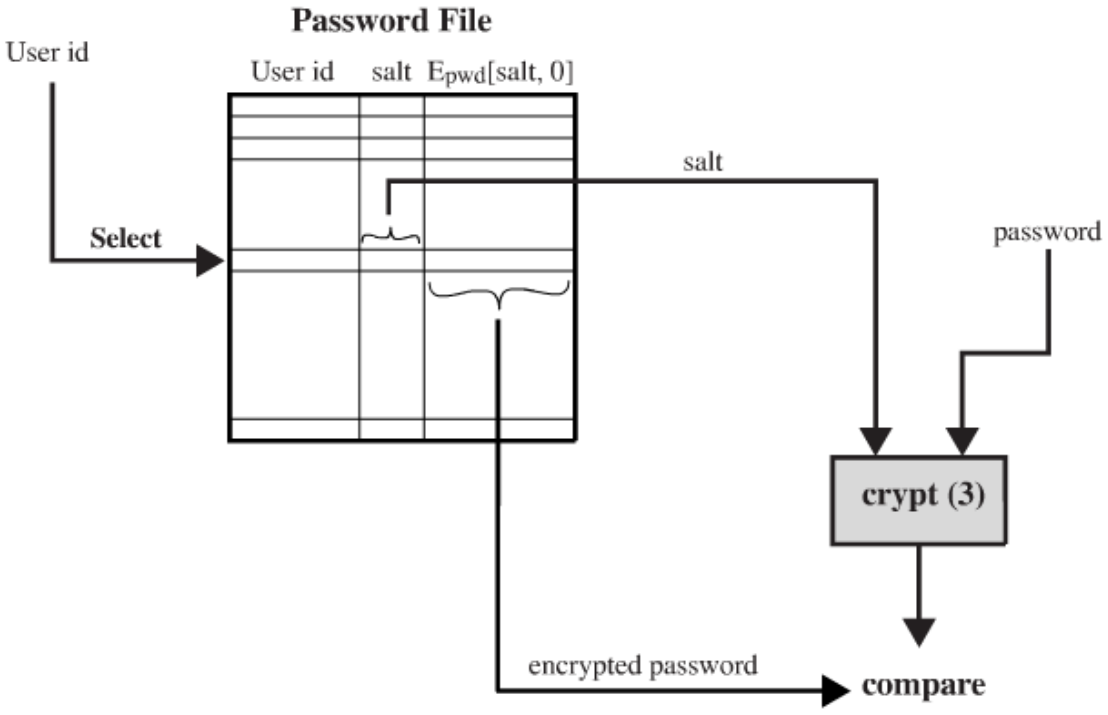
rw- r-- r-- root root *group* groups defined and their users

Permissions of the executable command:






**(a) Loading a new password**




**(b) Verifying a password**

Notes:

In addition to the classic *Data Encryption Standard (DES)*, there is an advanced symmetric-key encryption algorithm *AES (Advanced Encryption Standard).* The AES-128, AES-192 and AES-256 use a 128-bit block size, with key sizes of 128, 192 and 256 bits, respectively

Most linux systems use Hash Functions for authentication: Common message-digest functions include MD5, which produces a 128-bit hash, and SHA-1, which outputs a 160-bit hash.

# SETUID system call

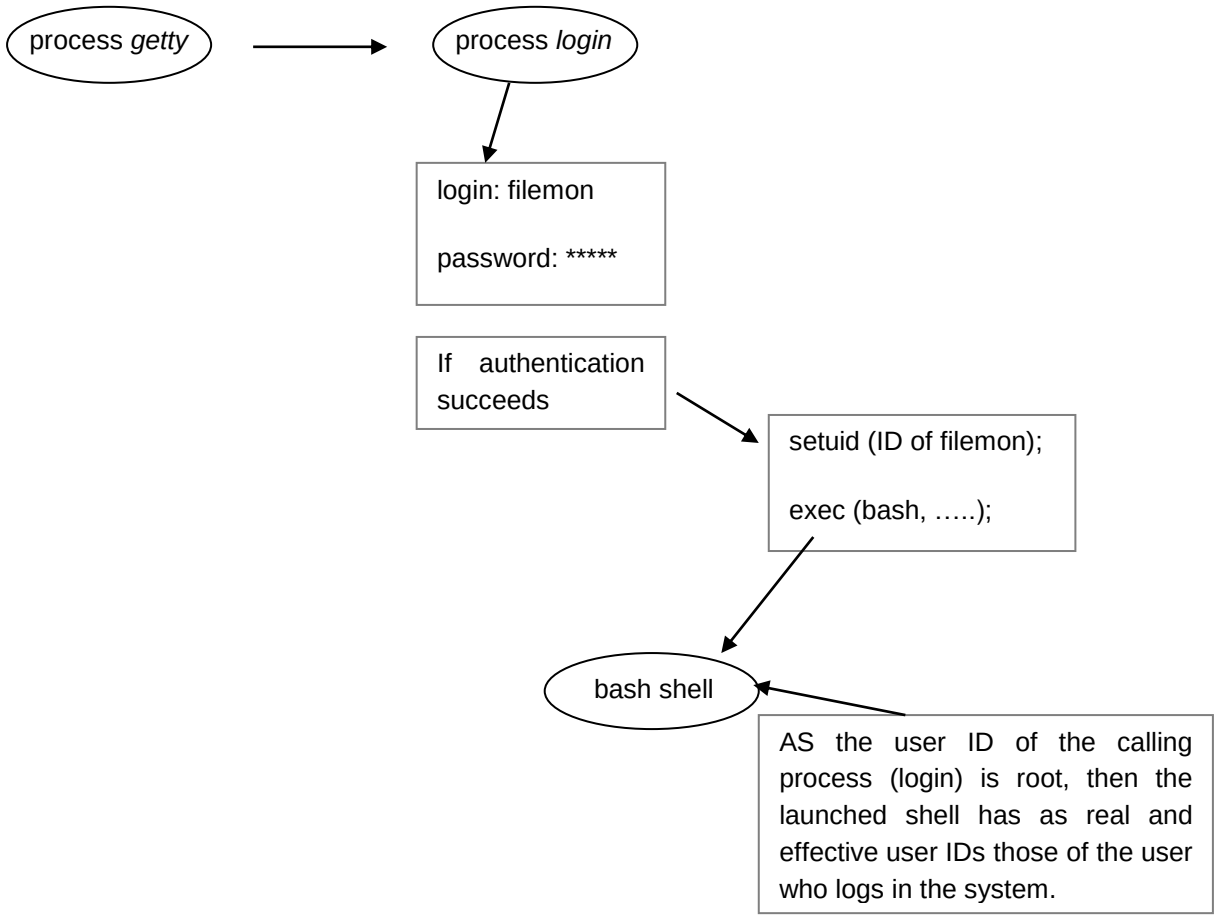
Syntax: *setuid (uid)*

*uid* is the new user ID. Its result depends on the current value of the effective used ID

The system call succeeds in the following cases:

1. If the effective user ID of the calling process is the superuser (root), the kernel sets as real and effective user ID the input parameter *uid*.
2. If the effective user ID of the calling process is not the superuser:
   2.1 If uid = real user ID, the effective user ID is set to uid (success).
   2.2 Else if *uid* = saved effective user ID, the effective user ID is set to *uid* (success).
   2.3 Else return error.

Example of case 1: *login* process



Example of case 2:

```
#include <fcntl.h>
main()
{
        int uid, euid, fdmjb, fdmaury;

        uid = getuid();                 /* get real UID */
        euid = geteuid();           /* get effective UID */
        printf("uid %d euid %d\n", uid, euid);

        fdmjb = open("mjb", O_RDONLY);
        fdmaury = open("maury", O_RDONLY);
        printf("fdmjb %d fdmaury %d\n", fdmjb, fdmaury);

        setuid(uid);
        printf("after setuid(%d): uid %d euid %d\n", uid, getuid(), geteuid());

        fdmjb = open("mjb", O_RDONLY);
        fdmaury = open("maury", O_RDONLY);
        printf("fdmjb %d fdmaury %d\n", fdmjb, fdmaury);

        setuid(euid);
        printf("after setuid(%d): uid %d euid %d\n", euid, getuid(), geteuid());
}
```

Example of Execution of Setuid Program

[Batch, 1986] Bach, M.J., The Design of the UNIX Operating System, Prentice-Hall, 1986.

Users: maury (ID 8319)
mjb (ID 5088)

Files:

| | | |
|---|---|---|
| maury | maury | r-- --- --- |
| Mjb | mjb | r-- --- --- |
| a.out | maury | rw**s** –x --x |

When "mjb" executes the file:

uid 5088 euid 8319

fdmjb -1 fdmaury 3

after setuid(5088): uid 5088 euid 5088

fdmjb 4 fdmaury -1

after setuid(8319): uid 5088 euid 8319

When "maury" executes the file:

uid 8319 euid 8319

fdmjb -1 fdmaury 3

after setuid(8319): uid 5088 euid 8319

fdmjb -1 fdmaury 4

after setuid(8319): uid 8319 euid 8319

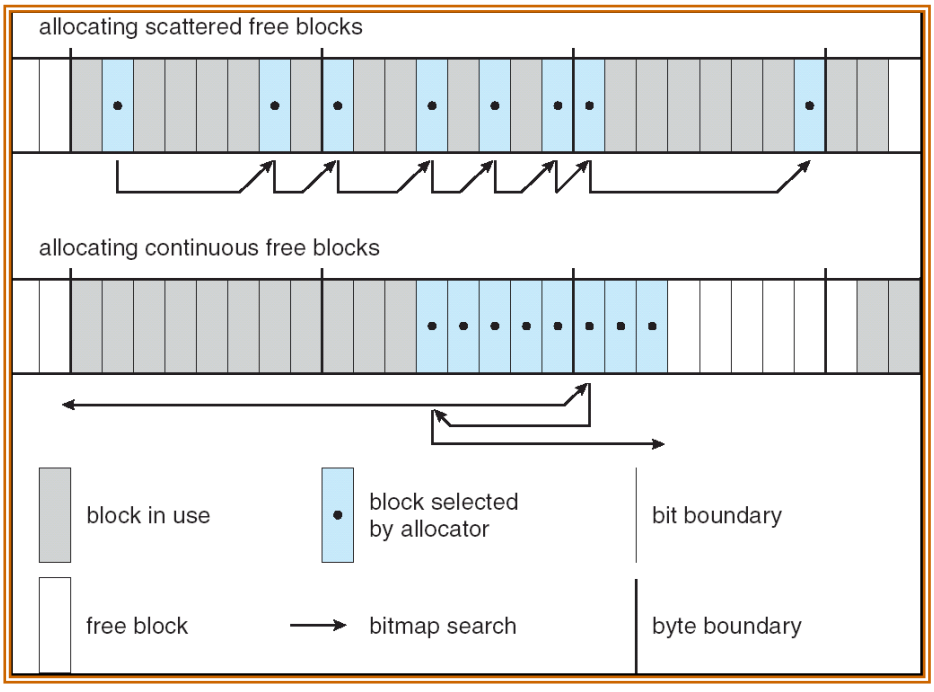
# 8. The Linux Ext2fs File System

Silberschatz, Galvin and Gagne ©2005
Operating System Concepts – 7th Edition,
Feb 6, 2005

- Ext2fs uses a mechanism similar to that of BSD Fast File System (ffs) for locating data blocks belonging to a specific file
- The main differences between ext2fs and ffs concern their disk allocation policies.
  - In ffs, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file.
  - Ext2fs does not use fragments; it performs its allocations in smaller units:

    The default block size on ext2fs is 1Kb, although 2Kb and 4Kb blocks are also supported.
  - Ext2fs uses allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation.

## Ext2fs Block-Allocation Policies



# 9. Journaling File Systems

- The system maintains a catching of file data and metadata (Buffer Cache).
- There can be inconsistencies in the file system due to a system crash of electric outage before the modified data in the cache (dirty buffers) have been written to disk.

  Related command: *fsck* (file system check)

- A journaling file system is a fault-resilient file system in which data integrity is ensured because updates to files' metadata are written to a serial log on disk before the original disk blocks are updated. The file system will write the actual data to the disk only after the write of the metadata to the log is complete. When a system crash occurs, the system recovery code will analyze the metadata log and try to clean up only those inconsistent files by replaying the log file.
- Linux file systems with journal: *ext3*, *ext4*, *ReiserFS*, *XFS* from SGI, *JFS* from IBM.

# Bibliography:

**[Batch, 1986]** Bach, M.J., *The Design of the UNIX Operating System*, Prentice-Hall, 1986.

**[Carretero y col., 2001]** Carretero Pérez, J., de Miguel Anasagasti, P., García Carballeira, F., Pérez Costoya, F., *Sistemas Operativos: Una Visión Aplicada*, McGraw-Hill, 2001.

**[Márquez, 2004]** Márquez, F.M., *UNIX. Programación Avanzada*, Ra-Ma, 2004.

**[Sánchez Prieto, 2005]** Sánchez Prieto, S., *Sistemas Operativos*, Servicio Public. Univ. Alcalá, 2005.

**[Silberschatz y col. 2005]** Silberschatz, A., Galvin, P. and Gagne, G., *Operating System Concepts* – 7th Edition, Feb 6, 2005.

**[Stallings 2005]** Stallings, W. *Operating Systems* (5th Edition), Prentice-Hall, 2005.

**[Tanenbaum 2003]** Tanenmaum, A., *Sistemas Operativos Modernos*, Prentice-Hall, 2003.