Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Part 3: File systems and I/O

Unit 8

Directories and protection

fSO

Goals and bibliography

- **Goals**
  - To know the concept of directory
  - To understand symbolic and hard links concepts
  - To know the techniques used to manage free disk space
  - To know the standard protection mechanism used in UNIX systems
- **Bibliography**
  - Silberschatz, chapters 10 and 11

Content

- **Directory concept**
- Directory implementation
- Links
- Free disk space management
- Protection

Directory concept

- **File system architecture: User sight**

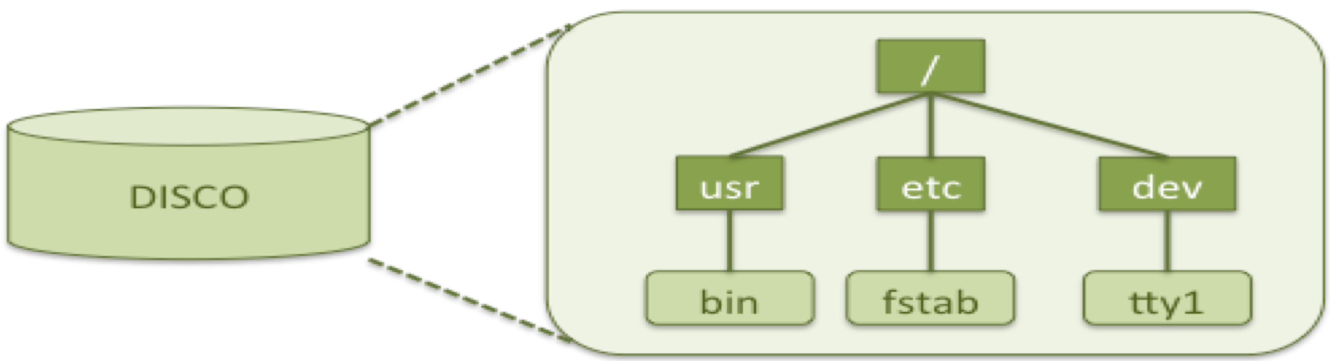
User level
File and directory abstractions

User libraries (to operate with files)
API with system calls related to files and directories

File operations:
- **Open/close**
- **Read/write**
- **Seek** within a file

Directory operations:
- **Create/remove**
- **Rename**
- Searching
- Navigating through the file system

Hierarchical view
Files and directories have a tree organization



Directory concept

- **A directory is a file**
  - An abstract data file: contains directory entries
  - It is the required element to organize files
- **Goals**
  - To find quickly a file from its associated name
  - To implement a convenient name scheme for users
  - To allow users to organize their files freely
  - To allow owners controlling operations permitted to other users on their files and directories

- **Operation on directories**
  - **Create entry (file or directory)**
    - It requires available space
  - **Remove entry**
    - It frees the disk space allocated to the entry and remove the corresponding directory entry
  - **Search by name**
    - It is performed sequentially
  - **List directory content**
    - It allows to see the directory entries inside the directory
  - **Rename entry**
    - It changes the name field in a directory entry
  - **Navigate the file system**
    - It allows accessing any point inside the directory hierarchy

Content

- Directory concept
- **Directory implementation**
- Links
- Free disk space management
- Protection

Directory implementation

- **Directory structure**

Directory -> keeps name to file associations

Flat

➢All files located inside a unique directory (CP/M)

- Name collision
- File grouping is not allowed

Hierarchical

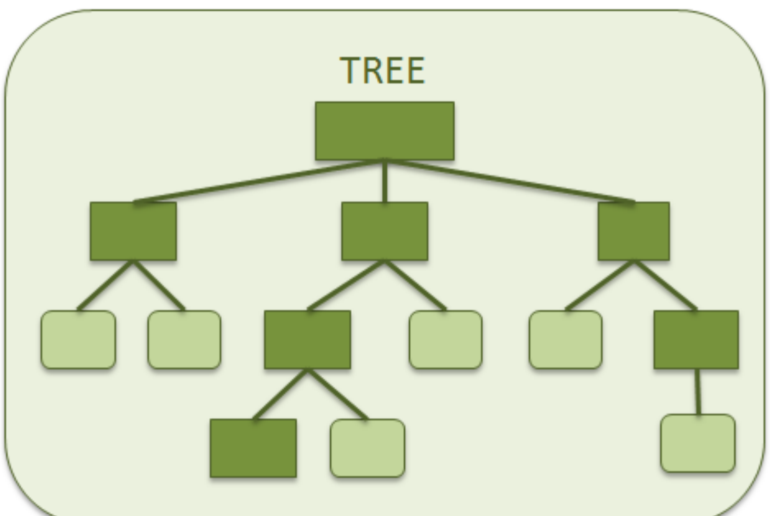
➢Organization in level (tree or graph) with any deep

- It allows arbitrary grouping
- It allows mounting/unmounting other file systems

• Hierarchical directory structure

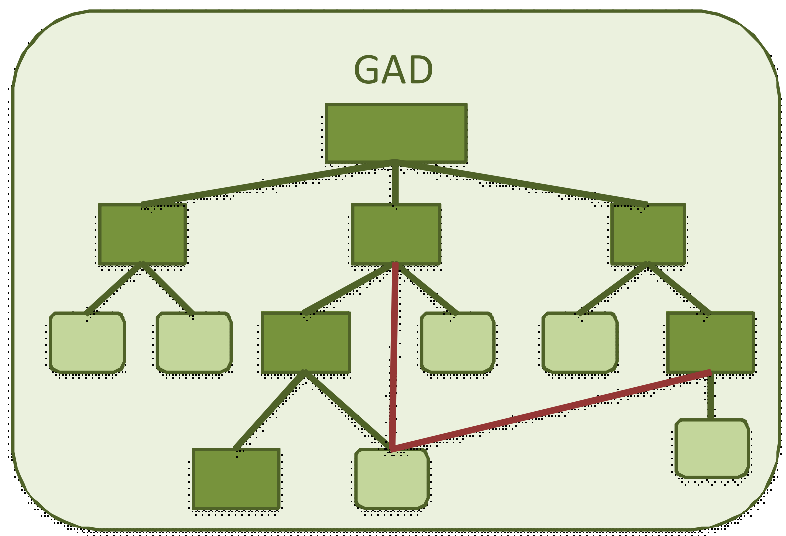
Tree

- Efficient search and grouping
- Only one name/location per entry
- Relative path = path from the actual directory to the entry



Direct acyclic graph

- Tree based structure, but it allows sharing files and directories
- Several names/paths for a given entry
- It doesn't allow cycles



Directory implementation

- **Directory content**
  - It is organized as registers named **directory entries**
  - It has a directory entry per element (file or directory) inside the directory
- **Directory entry** Its format depends on the file system
  - UNX: Name + i-node reference
  - Windows: Name + attributes + reference to data
- **Directory location** in disk
  - Centralized in a disk dedicated area (flat)
  - In files (hierarchical)

Directory implementation

- **Directories in FAT (12, 16)**
  - Root directory of fixed size and located in a dedicated area
  - All the other directories are managed as files containing directory entries
  - A directory entry has 32 byte size
    - Name (8) + extension (3)
    - Attributes (i.e. date, time, size, etc.)
    - First data block index (FAT entry)

- **Directories in UNIX systems**
  - Directories are implemented as a **file type**
  - Data in a directory are structured as a table with two columns: i-node number and name

Directory entry

i-node number	File name

- Every directory corresponds to a file
- Entry named "." corresponds to the working directory (points to itself)
- Entry named ".." corresponds to the parent directory
- i-node number = index to find the i-node in the i-node vector stored in the disk

Root directory entry

i-node	File name
1	.
1	..
3	dev
4	bin

Implementación de directorios

- **Acces by name mechanism**
  - In the disk there is a dedicated part to store the **i-nodes vector**
    - i-node = attributes + file data localization
    - In the i-nodes vector there is an i-node per file
  - Absolute name:
    - File search starts in the root directory
    - The root directory has a fixed i-node (i-node 1 in MINIX)
    - Example: /a/b/c
  - Relative name
    - File search starts in the working directory
    - Example: b/c

```
Whilre elements remain
  If it is a directory
    Check permissions,
    Localize element in the directory,
    Get the i-node
Return final i-node
```

i-node	File name
1	.
1	..
3	dev
4	bin
6	a

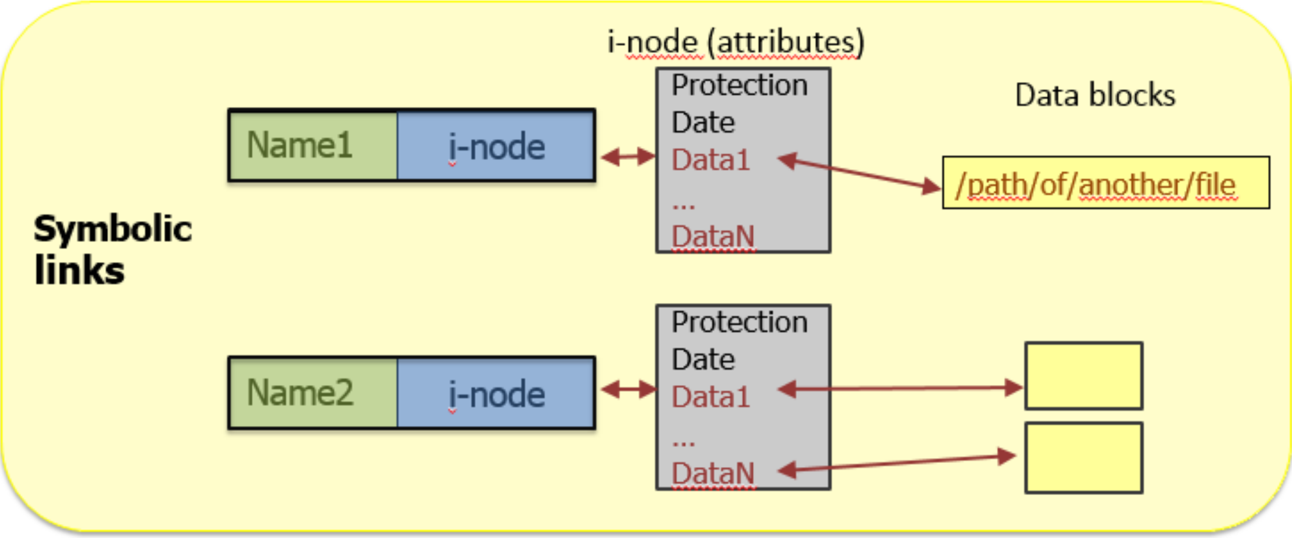
i-node	File name
6	.
1	..
10	b

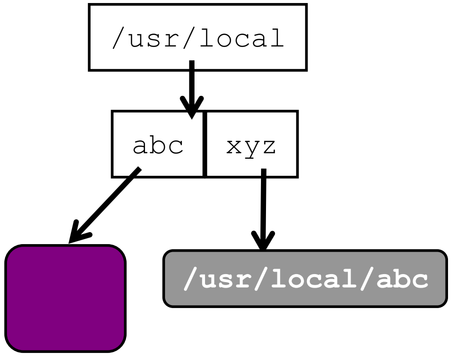
i-node	File name
10	.
6	..
20	c

Content

- Directory concept
- Directory implementation
- **Links**
- Free disk space management
- Protection

- **Symbolic links**
  - UNIX terminology (Windows 7 also supports them by means of **mklink** command)
  - A is a symbolic link file that links with file B
    - One A's attribute indicates that it is a link
    - The OS interprets A data as a path to access file B
    - The OS redirects read and write operations on A to B
    - The access permissions that apply are the ones from B
  - B can be located in another file system from A (i.e. remotely mounted)
  - What happens if file B is deleted or moved to another location?
    - In some systems (i.e. MacOS) the OS itself corrects the link path
    - In other systems (i.e. Linux) the link becomes orphan and no longer works
  - Deleting A doesn't affect to B

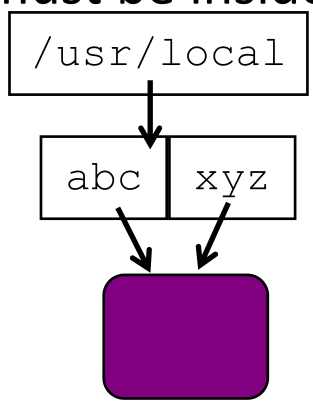




Symbolic links

• Hard links

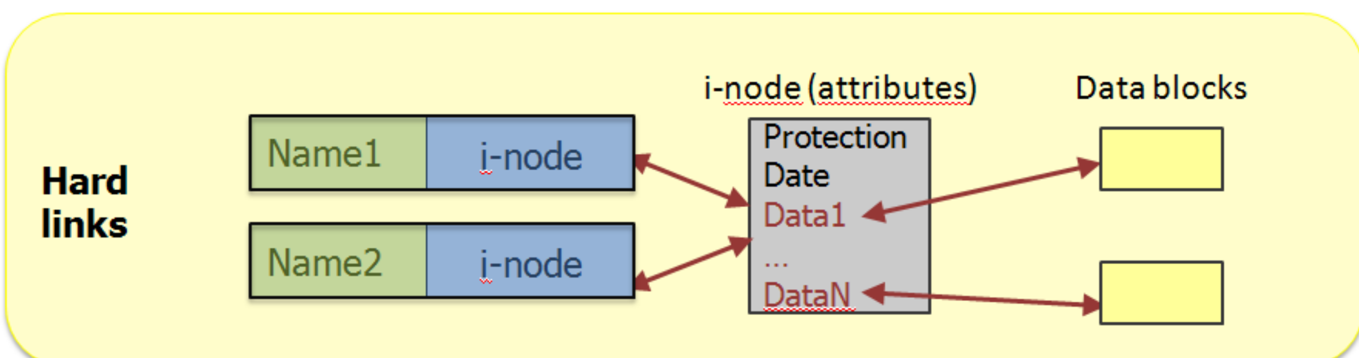
- Two or more directory entries contain the same i-node number
  - A file can be accessed from several paths (names)
- Every i-node keeps a counter with the number of directory entry that reference to it
  - The file (data + i-node) is deleted only when its last reference is deleted
- Hard links must be inside the same file system



i-node	File name
6	.
1	..
10	Name 1

i-node	File name
20	.
5	..
10	Name 2

i-node 10
….
N links = 2
….
Tripple ind point

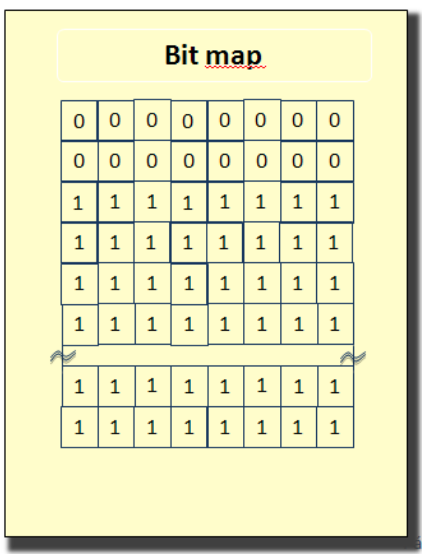


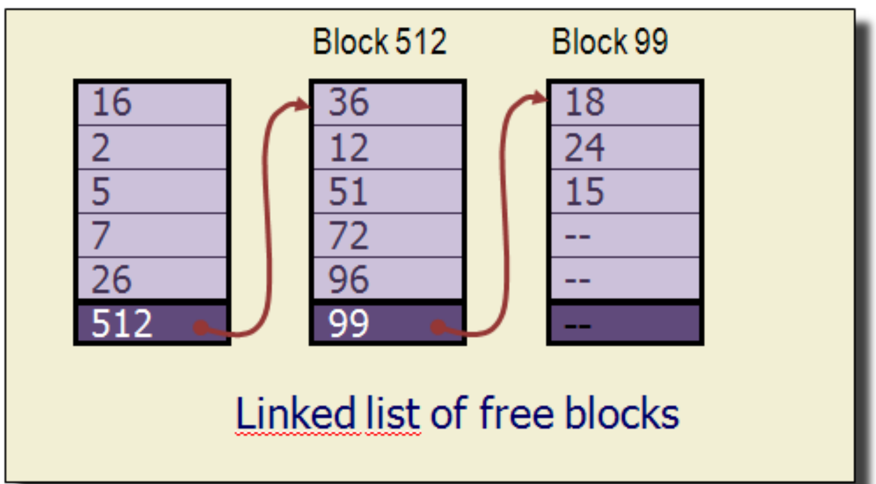
Content

- Directory concept
- Directory implementation
- Links
- **Free disk space management**
- Protection

Free disk space management

- Disk space is seen as a **blocks vector**
- At every moment the OS has to be able to know which ones are free
  - Any block doesn't work
    - Contiguity is required for efficiency sake
  - **Bit map**
    - The state for every block is represented by one bit (i.e. 1 value means free)
    - It is stored in a dedicated disk area
    - It allows efficient searching of consecutive blocks
  - **Linked list**
    - In a specific disk location it is maintained the index to the first free block
    - Every free block point to the next
  - **Grouping**
    - Free blocks are represented by means of a index block list
    - It easies inserting/extracting blocks, but it is difficult to look for contiguity

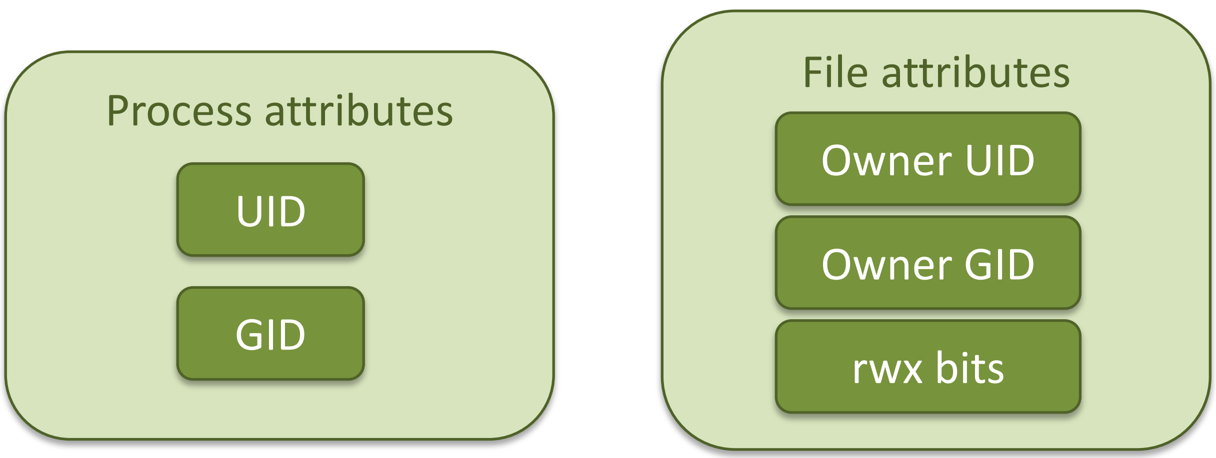




Content

- Directory concept
- Directory implementation
- Links
- Free disk space management
- **Protection**

- **Protection concept**
  - Mechanism used to control process access to system resources
- **How is protection implemented in UNIX systems?**
  - It is based on comparing process attributes with file attributes and then determining if the operation is allowed



- **Process protection attributes**
  - **User identifier**
    - Real UID (rUID) identifier of the process creator user
    - Effective UID (eUID) identifier of the process executable file owner
  - **Group identifier**
    - Real GID (rGID) identifier of the rUID user group
    - Effective GID (eGID) identifier of the eUID user group
- **File protection attributes**
  - **Permission bits:** 9 permission bits organized in three sets: owner, group and others
  - **Sample formats:** rwxr_xr_x, 0755, 04755, rwsr_xr_x
  - Interpretation:
    - **Regular files**: read, write and execution
    - **Directories:** list content (r), create or remove entries (w), move to subdirectories (x)
    - **Special:** read and write
  - SETUID and SETGID bits

- Attributes assignment
  - A file receives its attributes from the process that creates it

```
ownerUID = UID          ownerGID = GID
```

  - The process receives its attributes thanks to the inheritance mechanism and the information stored in /etc/passwd

```
Name:password:UID:GID:description:HOME:shell
```

  - A process can change its UID and GID when calling to exec() on a fie with SETUID or SETGID set
    - If the executable file has its SETUID bit set then the eUID becomes the file "ownerUID"
    - If the executable file has its SETGID bit set then the eGID becomes the file "ownerGID"
    - **Example**

-rwsr—r-x	1	felip	users	17	Jan	29	09:34	arxi1
-rwxr-sr-x	1	felip	users	223	Jan	29	09:34	arxi2

- **UNIX protection rules**

  When a process tries accessing a file the following rule sequence is applied:

  1) If process eUID=0 (root), no checking is performed and the file access is allowed.
     → Except execution permission that must be set at least for one domain
  2) If process eUID is the same as the file owner then the owner permissions are checked.
  3) Else if process eGID is the same as the file owner group then the group permissions are checked.
  4) Otherwise the other permissions are checked

→ Notice that after checking one permission set no others are checked

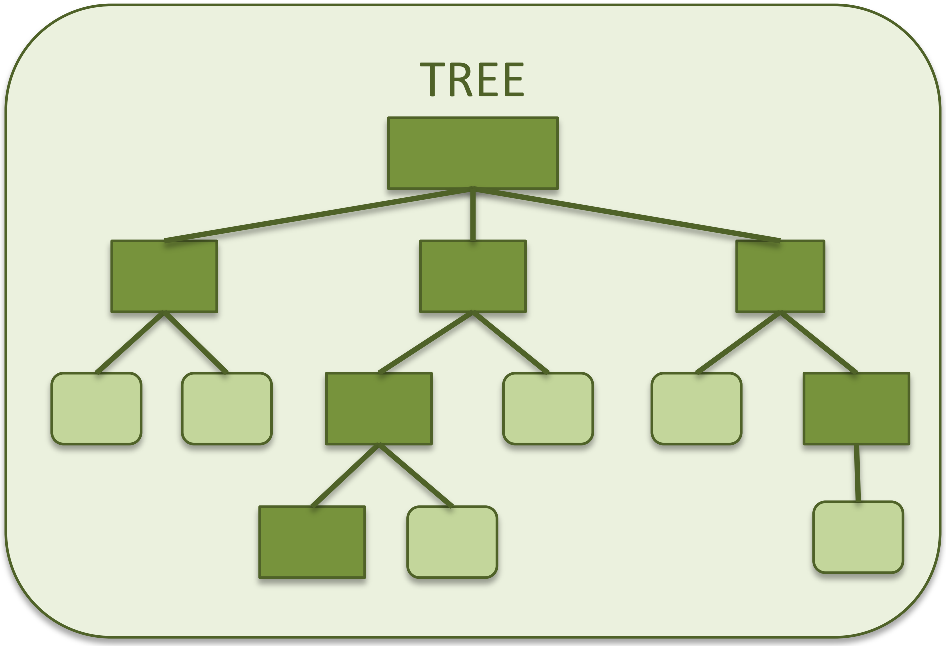
```
if UID = 0
then
   permission granted
else
   if eUID = ownerUID
   then
      permission granted
      according to user set
                   rws rwx r-x
   else
      if eGID = ownerGID
      then
        permission granted
        according to group set
                   rws rwx r-x
      else
        permission granted
        according to other set
                   rws rwx r-x
```

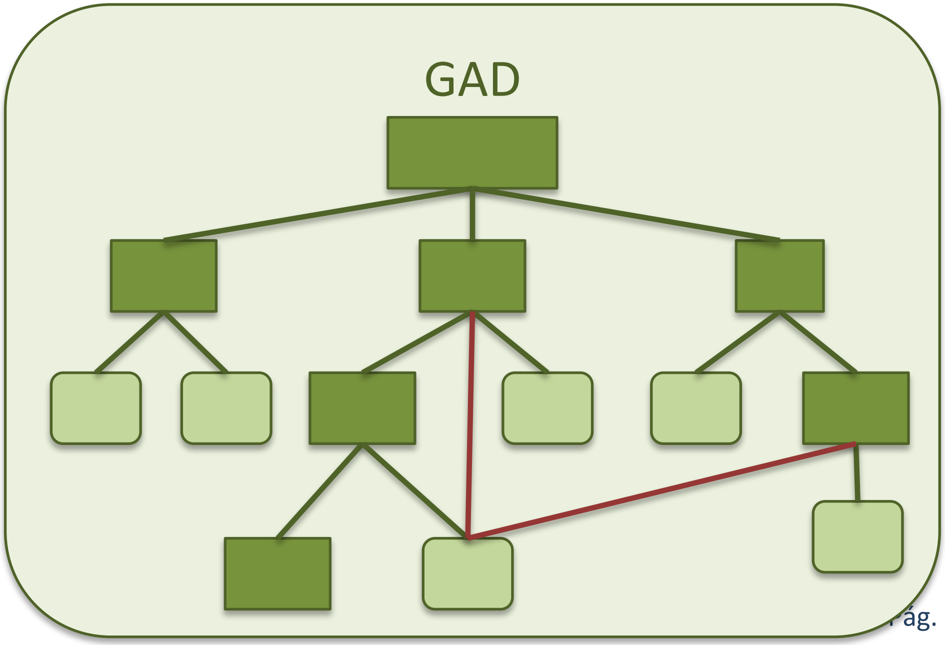
- **Example:**
- $ ls -l

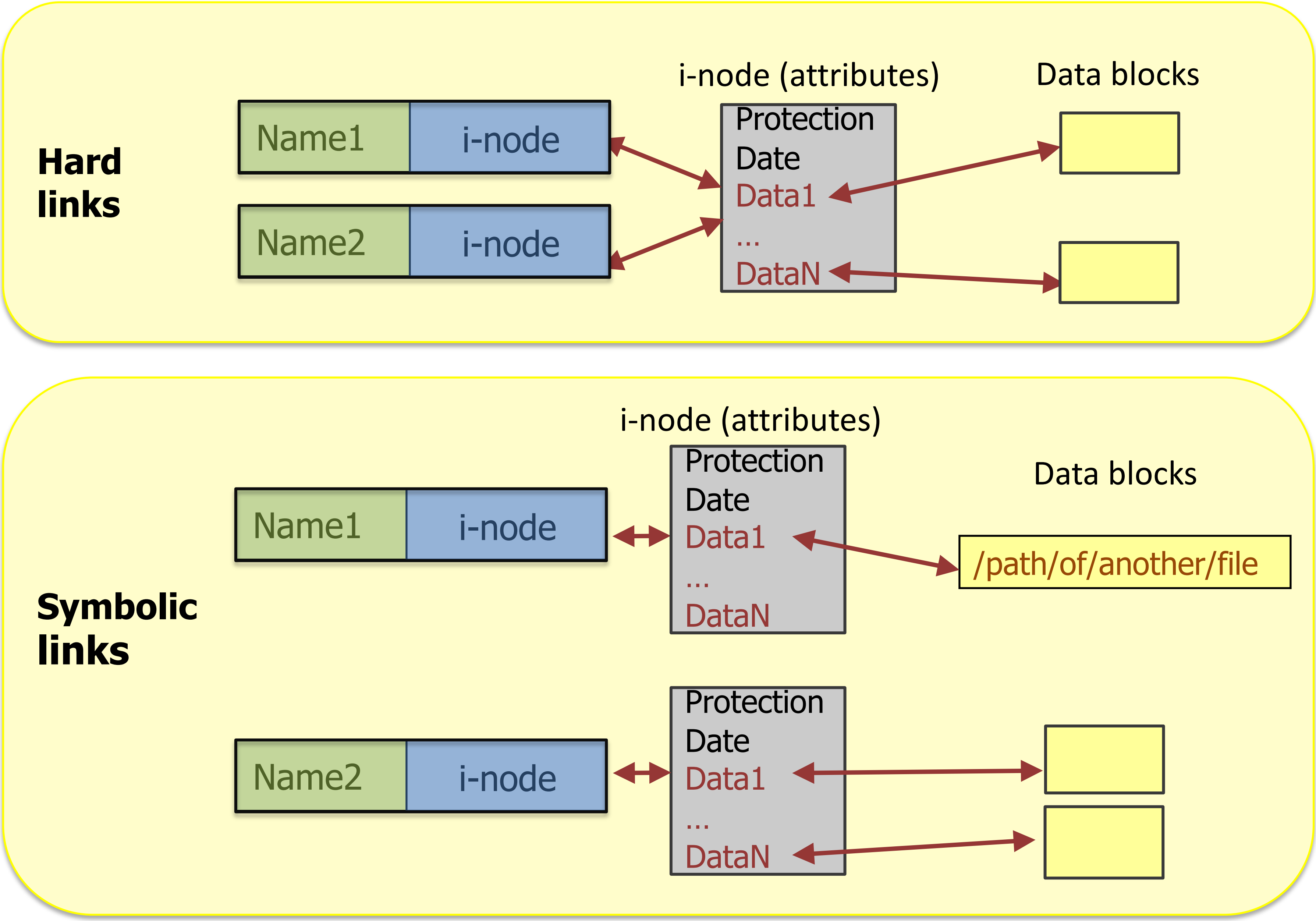
```
total 7
-rwsr-xr-x   1 felip users      17 Jan 29 09:34 ejec1
-rwxr-sr-x   1 felip users     223 Jan 29 11:03 ejec2
-rw-------   1 felip users    5120 Jan 29 12:00 datos
```

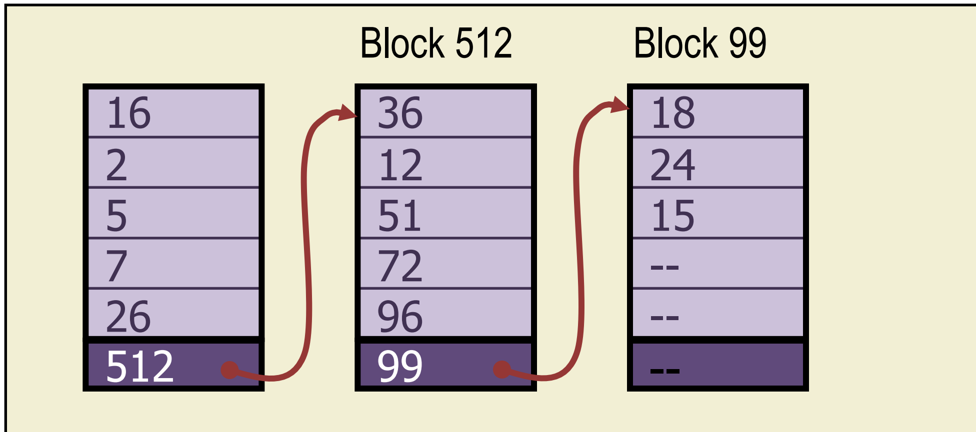
  - File **datos** can only be accessed (read and write) by processes started by user **felip**
  - Executable file **ejec1** has its SETUID bit set (look at **s** in owner permissions) and execution permission for group and other. That allows that when another user will start **ejec1** the process created will have **felip** as eUID and then it will be able to read and write into file **datos**

ANEXO - Dibujos






Hard links
Name1
i-node
Name2
i-node
i-node (attributes)
Protection
Date
Data1
…
DataN
Data blocks
Symbolic links
Name1
i-node
i-node (attributes)
Protection
Date
Data1
…
DataN
Data blocks
/path/of/another/file
Name2
i-node
Protection
Date
Data1
…
DataN



Linked list of free blocks