



OPERATING SYSTEMS:

Lesson 12: Directories

Jesús Carretero Pérez
David Expósito Singh
José Daniel García Sánchez
Francisco Javier García Blas
Florin Isaila



Goals

- To know the concepts of file and directory and their characteristics.
- To use file and directory management services offered by the operating system.
- To understand a file system structure.
- To understand the mechanisms supporting a file server and to apply them to simple exercises.



- **Directories**
- Structure alternatives
- Name interpretation
- Directory handling.



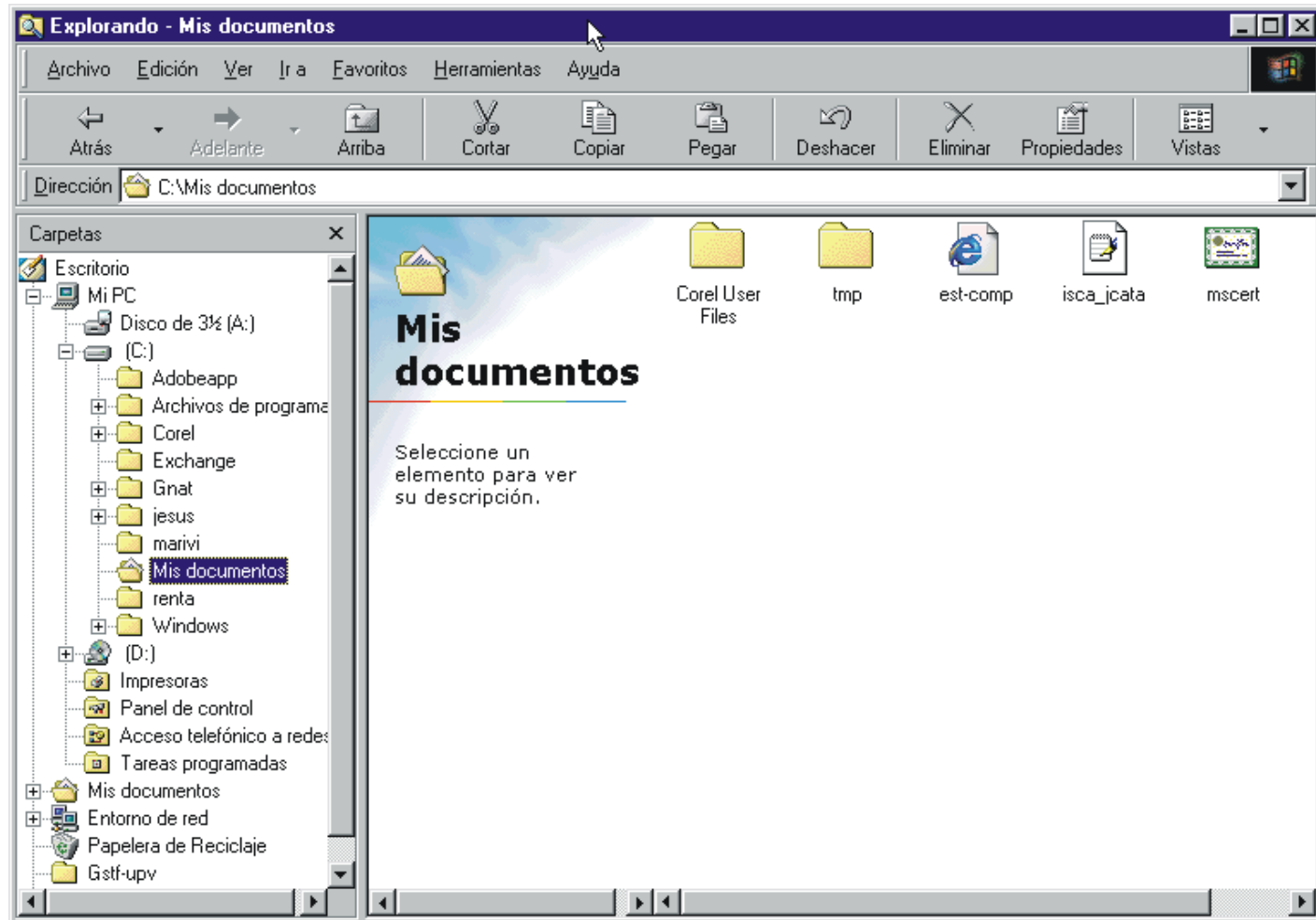
File organization

- A file system may store a great number of files.
- Mechanism needed to organize and locate files.
 - **Extensions:** Organization by file type.
 - **Directory:** abstraction of a file container.



- **Directory:**
 - Object uniquely relating a user file name with an internal file descriptor.
- Organizes and provides information about the file system structure.
- A directory has an entry per stored file.
- Entry information:
 - Internal file descriptor.
 - In some cases, some file attributes.

Example: Windows explorer





Directories: logical view

- Hierarchical approach.
- When a file is opened the OS looks for the name in the directory structure.
- Operations on directories:
 - Create and erase directories.
 - Open and close directories.
 - Rename directory.
 - Read directory entries.
- Directory hierarchical organization:
 - Simplifies file naming (unique names).
 - Provides distribution management => group files logically (same user, same application, same task, ...)



- Directories
- **Structure alternatives**
- Name interpretation
- Directory handling.

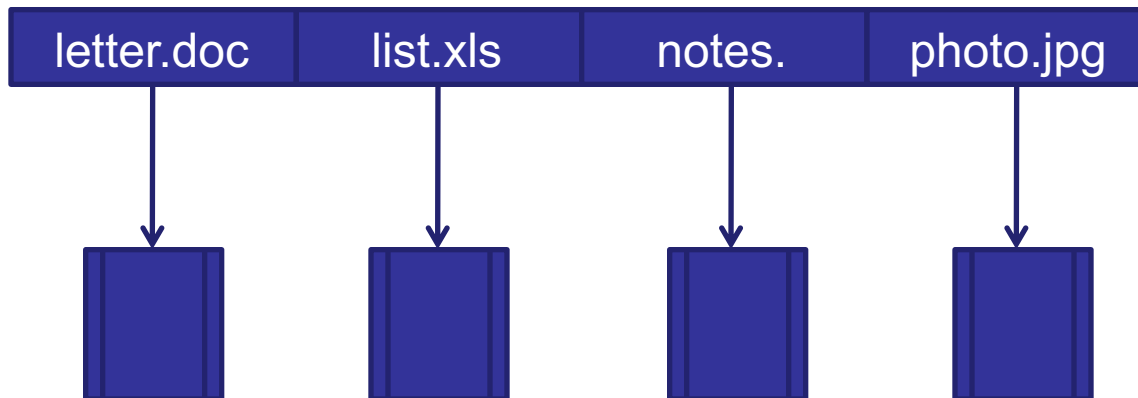


Structure alternatives

- Single level directory.
- Two-levels directory.
- Tree structure directory.
- DAG (directed acyclic graph) structure directory.
- Generalized graph structure directory.

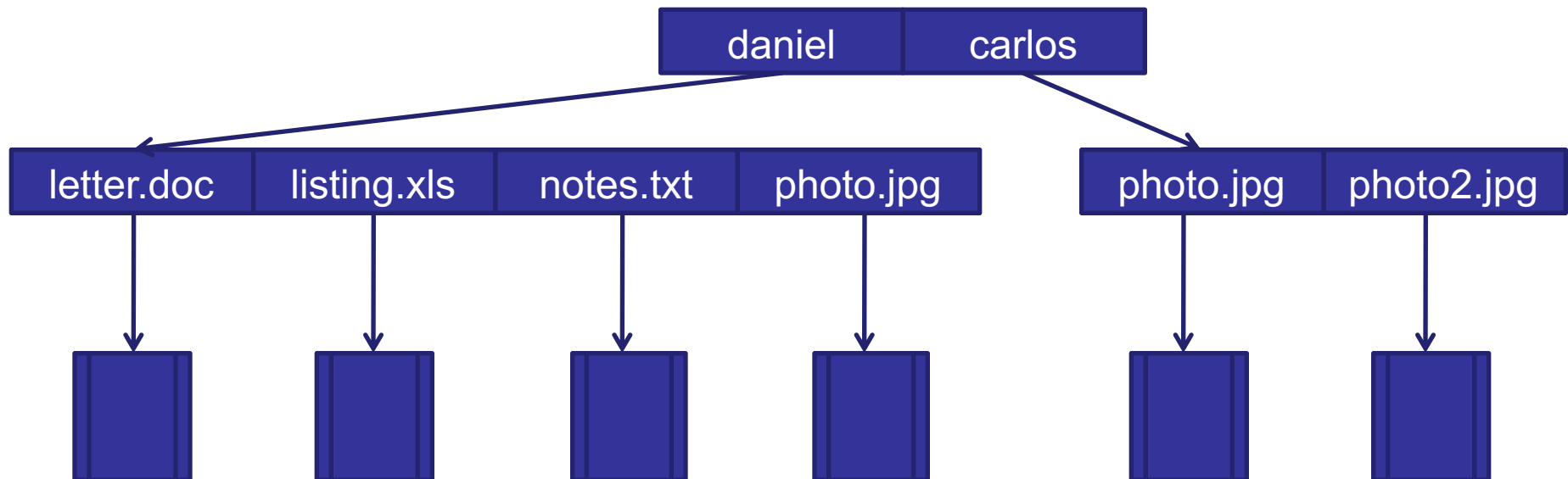
Single level directory

- A single directory for all users.
- Problems with file naming.
 - High probability of name coincidence.



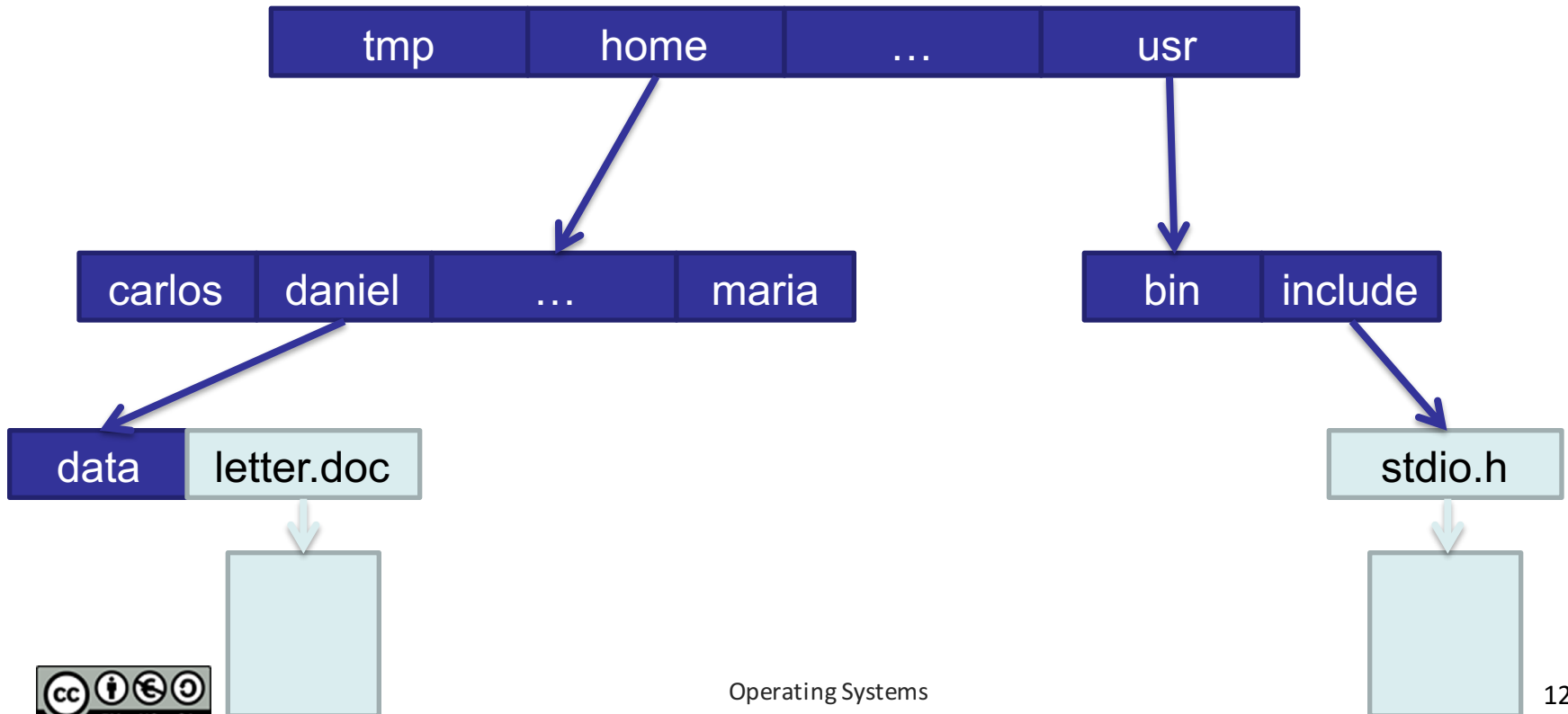
Two-levels directory

- A directory per user.
- Automated or manual path.
- Same file name for multiple users is valid.
- Efficient lookup, but grouping problems.



Tree structure directory

- Efficient lookup.
- Relative and absolute naming -> working directory.



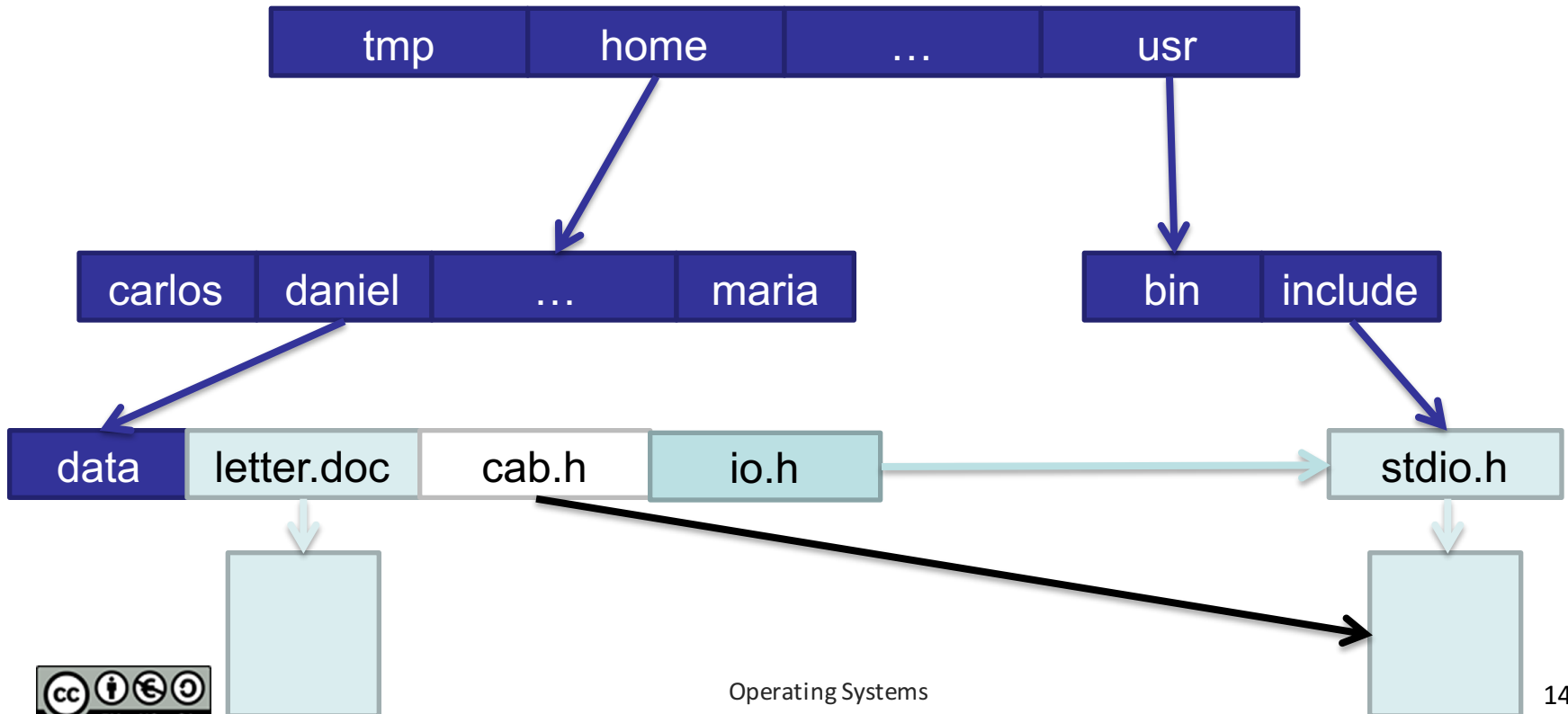


Tree structure directory

- Absolute names contain the full path.
- Relative names start from working (or current) directory.
- Directory change:
`cd /spell/mail/prog`
`cd prog`
- Erase file: **rm** <filename>
- Create directory: **mkdir** <dir-name>
- Example:
`cd /spell/mail`
`mkdir count`
`ls /spell/mail/count`
- Erase directory: **rm -r** mail

Acyclic graph structure directory

- Has shared files and directories.
- Concept not visible to Windows users.





Acyclic graph directory

- **link:** A file with multiple names -> link control
 - A single file with link count in descriptor (physical link).
 - New file type with target name in file content (symbolic link).
- Link removal:
 - A. Decrement counter; if 0 erase file.
 - B. Traverse links and erase all.
 - C. Erase only link and leave the rest.
- Problem: close loops traversing the tree.
- Solutions:
 - Allow only links to files, but not for directories.
 - Loop detection algorithm when link is created.
- UNIX implementation limitation: physical links only within same file system.



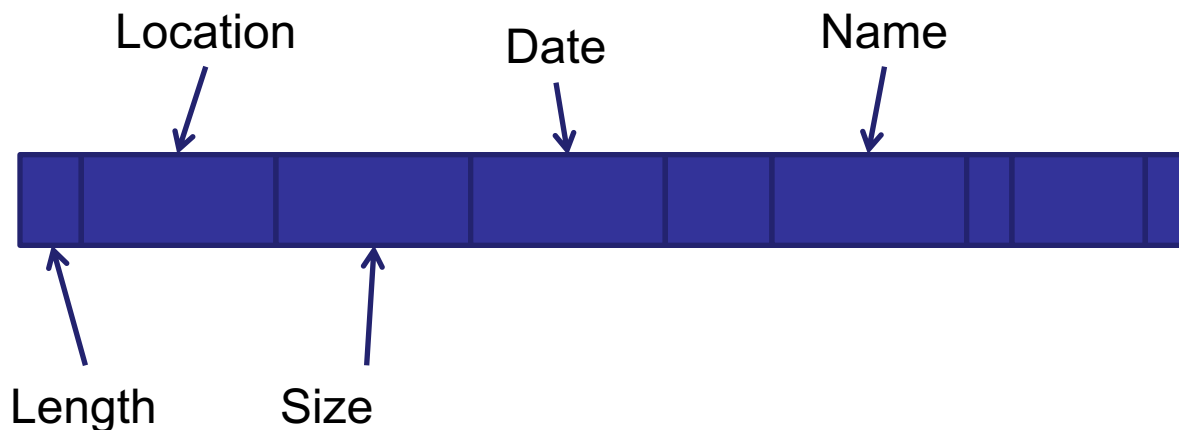
Directory structure

- Directory structure and files stored in disks.
- Implementation alternatives for directories:
 - Use special block with directory information.
 - Use file whose content is the directory.
- Information in directory: name, type, address, max and current length, access and modification time, owner, ...
 - In case of using a file, most of the contents are file metadata.



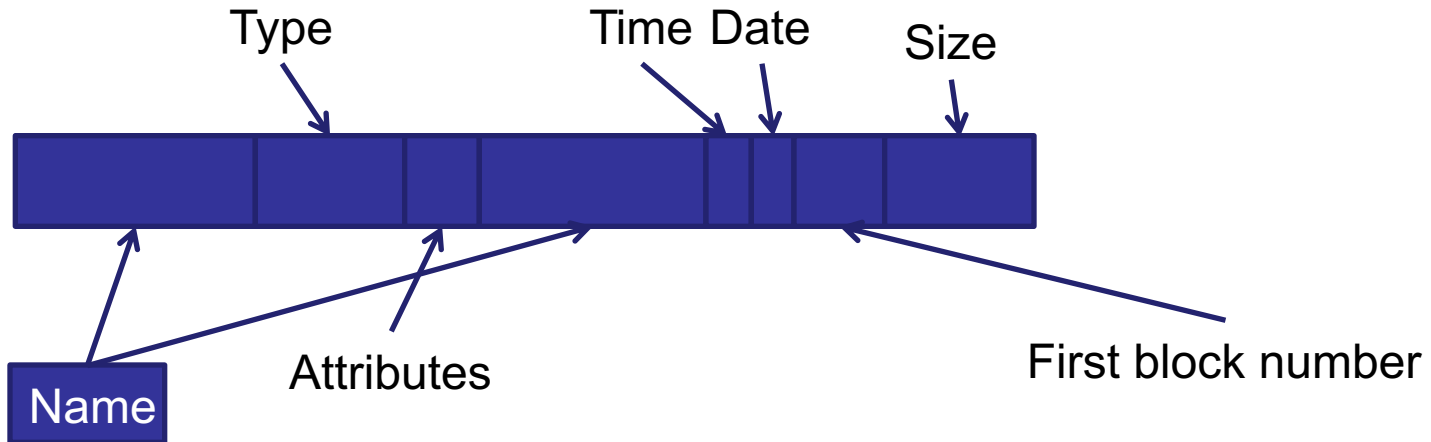
- Directories for contiguous files.
 - Assume all files are stored with contiguous allocation.
- Directories for linked files.
 - Assume all files are stored with non-contiguous allocation and blocks are represented as a linked list.
- Directories for indexed files.
 - Assume all files are stored with non-contiguous allocation and blocks or extents are represented through an indexed structure.

- Directory entry:
 - File attributes in directory entry.
 - File first block identifier.
 - File size.
- Example: ISO-9660 format for CD-ROM



Directory for linked files

- Directory entry:
 - File attributes.
 - First block number.
 - File size.
- Example: FAT





Directory for indexed files

- Most popular alternative
- Directory entry:
 - Name.
 - Metadata identifier for file (i-node, MFT entry, ...)

| i-node id | Name |
|-----------|------|
|-----------|------|



- Advantages:
 - No need to modify directory to change file attributes.
 - No need to modify directory when file changes its length.
 - An i-node may represent a directory or a file.
 - Simplified construction of hierarchical systems.
 - Name length is not prefixed.
 - Easy creation of name synonyms (links) for a file name.



Directory organization

- Efficiency: fast file lookup.
- Naming: Convenient and easy for users.
 - Two users may have the same name for different files.
 - Same file may have different names.
 - Variable length names.
- Grouping:
 - Logical grouping for files according to properties (e.g. c++ programs, games, ...)
- Structuring:
 - Access operations clearly defined and structure hiding.
- Simplification:
 - Directory entry must be as simple as possible.



- Absolute name: Path from root directory (/ in GNU/Linux, \ in Windows).
- Relative name Path from directory different from root.
 - Example: (you are in /users/) daniel/keys
 - Relative to the working directory (*pwd*)
- Special directories:
 - . Working directory. Example: cp /users/daniel/keys .
 - .. Parent directory. Example: ls ..
 - HOME: Base directory for a given user.



- Directories
- Structure alternatives
- **Name interpretation**
- Directory handling.



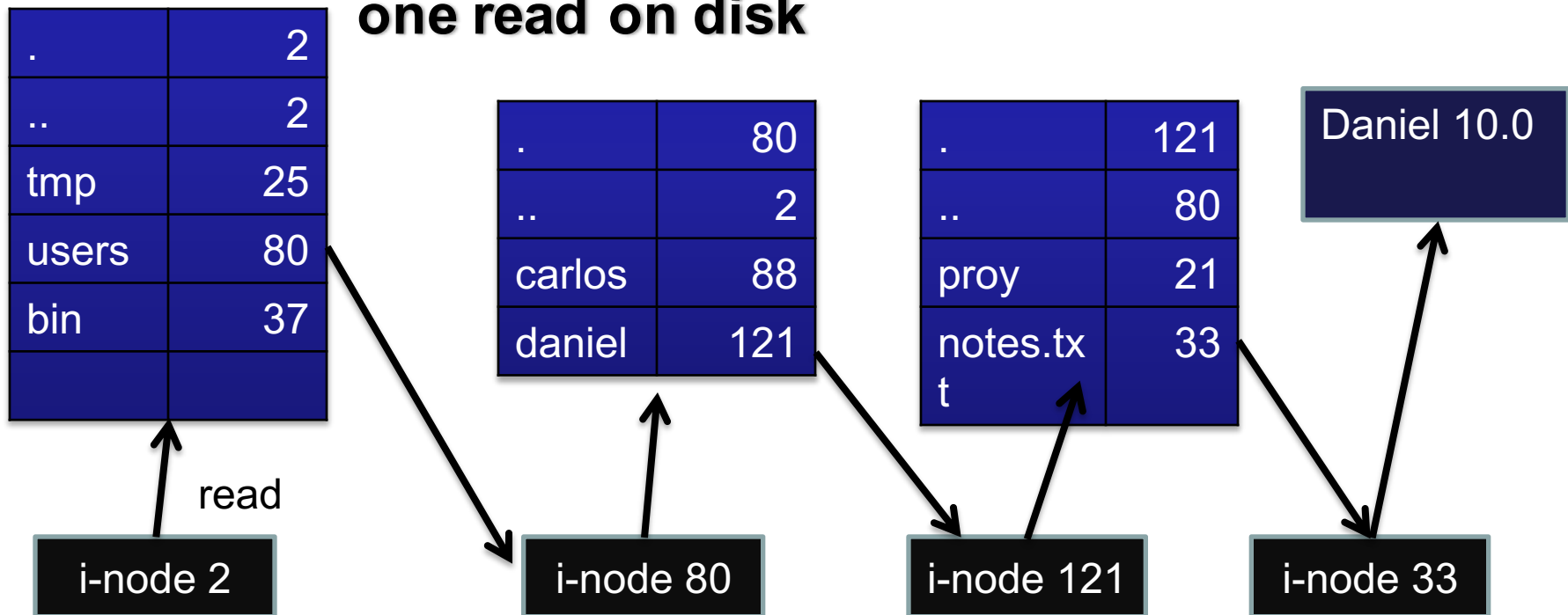
Name interpretation in UNIX

- Each directory is stored as a file with pairs <i-node number, file-name>.
- Initially in memory directory for /.
- How many disk blocks does a directory need for storage?
 - Depends on the number of files in directory and the length of names.
- Lookup in directory is sequential.

Name interpretation in UNIX

- Find i-node for file `/users/daniel/notes.txt`.

Directory traversing many lead to more than one read on disk





Directory hierarchy

- Single directory tree?
 - Per logical device in Windows (c:\users\carlos\keys, j:\joe\tmp, ...)
 - System wide in UNIX (/users/carlos/keys, /joe/tmp, ...)
- Services for building hierarchy are needed:
mount and **umount**.
 - `mount /dev/hda /users`
 - `umount /users`
- **Advantages:**
 - Single system image and hiding device type.
- **Drawbacks:**
 - More complex name translation.
 - Problems with physical file links.

- Volume:
 - Set of coherent metainformation and data.

FAT



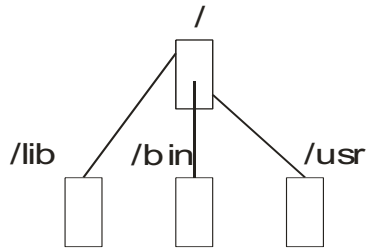
UNIX



Mounting partitions

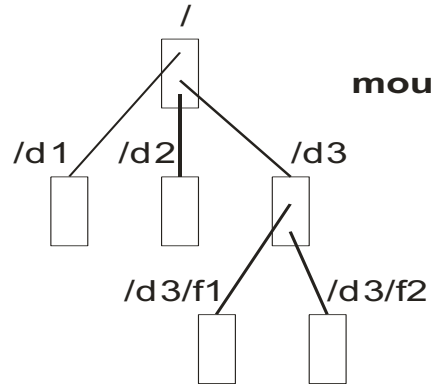
Root volume

Volumen raiz
(/dev/hd0)

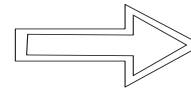


Un-mounted volume

Volumen sin montar
(/dev/hd1)

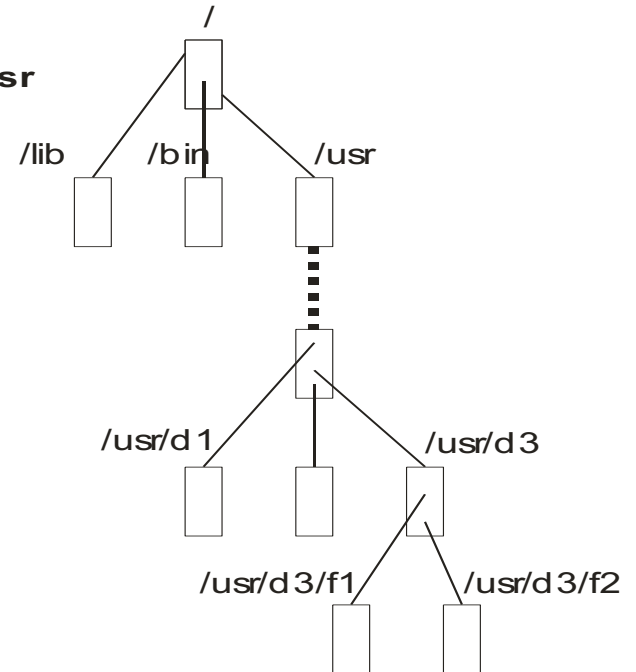


mount /dev/hd1 /usr



Mounted volume

Volumen montado





- Directories
- Structure alternatives
- Name interpretation
- **Directory handling.**



Example: directory handling

- Services for handling files representing directories.
- How does one know if a name corresponds to a file or a directory?
- Service:

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int stat(char *name, struct stat *buf);
```

```
int fstat(int fd, struct stat *buf);
```

```
...
```

```
cond = S_ISDIR(buf.st_mode) /* true for directories*/
```



Mkdir – Create directory

- Service:

```
#include <sys/types.h>
```

```
#include <dirent.h>
```

```
int mkdir(const char *name, mode_t mode);
```

- Arguments:

- **name**: directory name.
- **mode**: protection bits.

- Returns:

- Zero or -1 on error.

- Description:

- Creates a directory named **name**.
- Owner UID = effective UID.
- Owner GID = effective GID.



Rmdir – Remove directory

- Service:

```
#include <sys/types.h>  
int rmdir(const char *name);
```

- Arguments:

- **name**: Directory name.

- Returns:

- Zero or -1 on error.

- Description:

- Remove directory if it is empty.
- Otherwise directory is not removed.



Opendir – Open a directory

- Service:

```
#include <sys/types.h>  
#include <dirent.h>  
DIR *opendir(char * name);
```

- Arguments:

- **dirname**: Directory name.

- Returns:

- A pointer to be used with **readdir()** or **closedir()**.
- NULL on error.

- Description:

- Opens a directory as a sequence of entries.
- Places pointer in first entry.



Closedir – Cerrar un directorio

- Service:

```
#include <sys/types.h>
```

```
#include <dirent.h>
```

```
int closedir(DIR *dirp);
```

- Arguments:

- **dirp**: Pointer returned by opendir().

- Returns:

- Zero or -1 if error.

- Description:

- Closes association between **dirp** and directory entry sequence.



Readdir – Read directory entries

- Service:

```
#include <sys/types.h>  
#include <dirent.h>  
struct dirent *readdir(DIR *dirp);
```

- Arguments:

- **dirp**: pointer returned by **opendir()**.

- Returns:

- A pointer to an object of type **struct dirent** representing directory.
- **NULL** on error.

- Description:

- Returns next entry in directory associated to dirp and advances pointer.
- Structure is implementation dependent but you can assume it has a member **char* d_name**.



- **Service:**

```
#include <sys/types.h>
```

```
#include <dirent.h>
```

```
void rewindir(DIR *dirp);
```

- **Arguments:**

- **dirp**: pointer returned by opendir().

- **Description:**

- Sets directory position pointer to the first entry.



Link – Create a directory entry

- Service:

```
#include <unistd.h>
```

```
int link(const char *existing, const char *new);
```

```
int symlink(const char *existing, const char *new);
```

- Arguments:

- **existing**: Name of existing file.
- **new**: name of new entry that will be linked to existing file.

- Returns:

- Zero or -1 if error.

- Description:

- Create a new physical or symbolic link to an existing file.
- The OS does not record which is the original file and which is the new one.



Unlink – Remove directory entry

- Service:

```
#include <sys/types>
```

```
int unlink(char *name);
```

- Arguments:

- **name**: File name.

- Returns:

- Zero or -1 if error.

- Description:

- Removes entry to directory and decrements number of links to file.
- When number of links equals zero and no process keeps it open, space is freed and file is no longer accessible.



- Service:

```
int chdir(char *name);
```

- Arguments:

- **name**: directory name

- Returns:

- Zero or -1 if error.

- Description:

- Modifies current directory used to form relative paths.



Rename – Change file name

- Service:
`#include <unistd.h>`
`int rename(char *old, char *new);`
- Arguments:
 - **old**: Name of existing file.
 - **new**: New file name.
- Returns:
 - Zero or -1 if error.
- Description:
 - Change name of file **old**.
 - New name is **new**.



- Service:

```
char *getcwd(char *buf, size_t size);
```

- Arguments:

- **buf**: pointer to buffer to store name of current directory.
- **size**: Length in byts of buffer.

- Returns:

- Pointer to **buf** or **NULL** if error.

- Description:

- Gets name of current directory.



Example: Directory listing

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

#define MAX_BUF 256

void main(int argc, char **argv) {
    DIR *dirp;
    struct dirent *dp;
    char buf[MAX_BUF];

    /* print current directory*/
    getcwd(buf, MAX_BUF);
    printf("Current directory: %s\n", buf);
}
```



Example: Directory listing

```
/* Open directory argument */  
dirp = opendir(argv[1]);  
  
if (dirp == NULL) {  
    fprintf(stderr, "Cannot open %s\n", argv[1]);  
}  
else {  
    /* read entry by entry*/  
    while ( (dp = readdir(dirp)) != NULL)  
        printf("%s\n", dp->d_name);  
    closedir(dirp);  
}  
exit(0);  
}
```



OPERATING SYSTEMS:

Lesson 12: Directories

Jesús Carretero Pérez
David Expósito Singh
José Daniel García Sánchez
Francisco Javier García Blas
Florin Isaila