Chapter 10: File-System Interface

- File Concept
- Access Methods
- Directory Structure
- File System Mounting
- File Sharing
- Protection

1. File Concept

- File
  - A named collection of related information
  - Recorded on contiguous logical address space
  - Mapped by the operating system onto non-volatile physical devices

- Types
  - Data
    - numeric
    - character
    - binary
  - Program

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
    - Packing might be used
- Complex Structures
  - Formatted document
  - Relocatable load file
- Who decides:
  - Operating system
  - Program

File Attributes

- **Name** – only information kept in human-readable form
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, and executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring

```
[mlee@cs mlee]$ ls -l Tmp
total 716660
-rwxr-xr-x    1 mlee     mlee          751 Jan 28 13:19 check
-rw-rw-r--    1 mlee     mlee          513 Feb  3 09:51 First.class
-rw-rw-r--    1 mlee     mlee          443 Feb  3 08:52 First.java
-rw-r--r--    1 mlee     mlee    732942336 Feb  4  2005 KNOPPIX_V3.7-2004-12-
   08-EN.iso
-rw-rw-r--    1 mlee     mlee        82944 Jan 27 11:21 user-centric v.1.doc
-rw-rw-r--    1 mlee     mlee        79360 Feb  2 19:16 user-centric v.2.doc
drwxrwxr-x    4 mlee     mlee         4096 Jan 13 17:08 Winter2005
-rw-rw-r--    1 mlee     mlee          662 Feb  3 09:51 Worker.class
[mlee@cs mlee]$
```

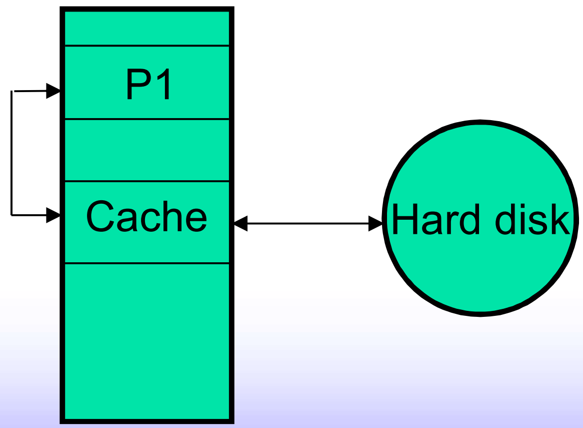
- Information about files are kept in the **directory** structure, which is maintained on the disk.

File Operations

- Create
- Write
- Read
- File seek – reposition within file
- Delete
- Truncate
- Open(F_i) – search the directory structure on disk for entry F_i, and move the content of entry to memory
- Close(F_i) – move the content of entry F_i in memory to directory structure on disk
- Append
- Rename
- Copy

Open Files

- Several pieces of data are needed to manage open files:
  - **Open-file table**: to avoid constant directory searching for read(), write(), …
  - **File pointer**: pointer to last read/write location, per process that has the file open
  - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
  - **Disk location of the file**: cache of data access information
  - **Access rights**: per-process access mode information



Open Files - continued

- Several pieces of data are needed to manage open files:
  - **File Descriptor Table (FDT)**
    - Array of pointers
    - 1 pointer for each file opened by that process
    - The pointers point to **Open File Table** entries
    - File descriptor
      - Index into the FDT
      - Used when read(), write(), close(), …
    - Note: first 3 entries of an FDT are special
      - **stdin**
      - **stdout**
      - **stderr**
    - 1 per process

Open Files - continued

- Several pieces of data are needed to manage open files:
  - **Open File Table (OFT)**
    - Array of entries
    - 1 entry per open file
    - If 2 processes open same file, 2 entries will be created
    - Each entry includes:
      - Pointer to an **Inode Table** entry
      - Byte offset
      - How the file is being used (e.g., reading, writing, both)
      - # of pointers from FDT's
    - 1 per system

Open Files - continued

- Several pieces of data are needed to manage open files:
  - **Inode Table (IT)**
    - Exactly 1 entry for each open file
    - Each entry includes:
      - File's **inode**
      - # of pointers to the entry from the Open File Table
    - **inode** = data structure (1 per file) containing
      - Uid, gid of file owner
      - Permission bits
      - 3 times
        - Last access
        - Last file mod
        - Last mod to i-node itself
      - Locations of file blocks on disk
    - 1 per system

Open Files - continued

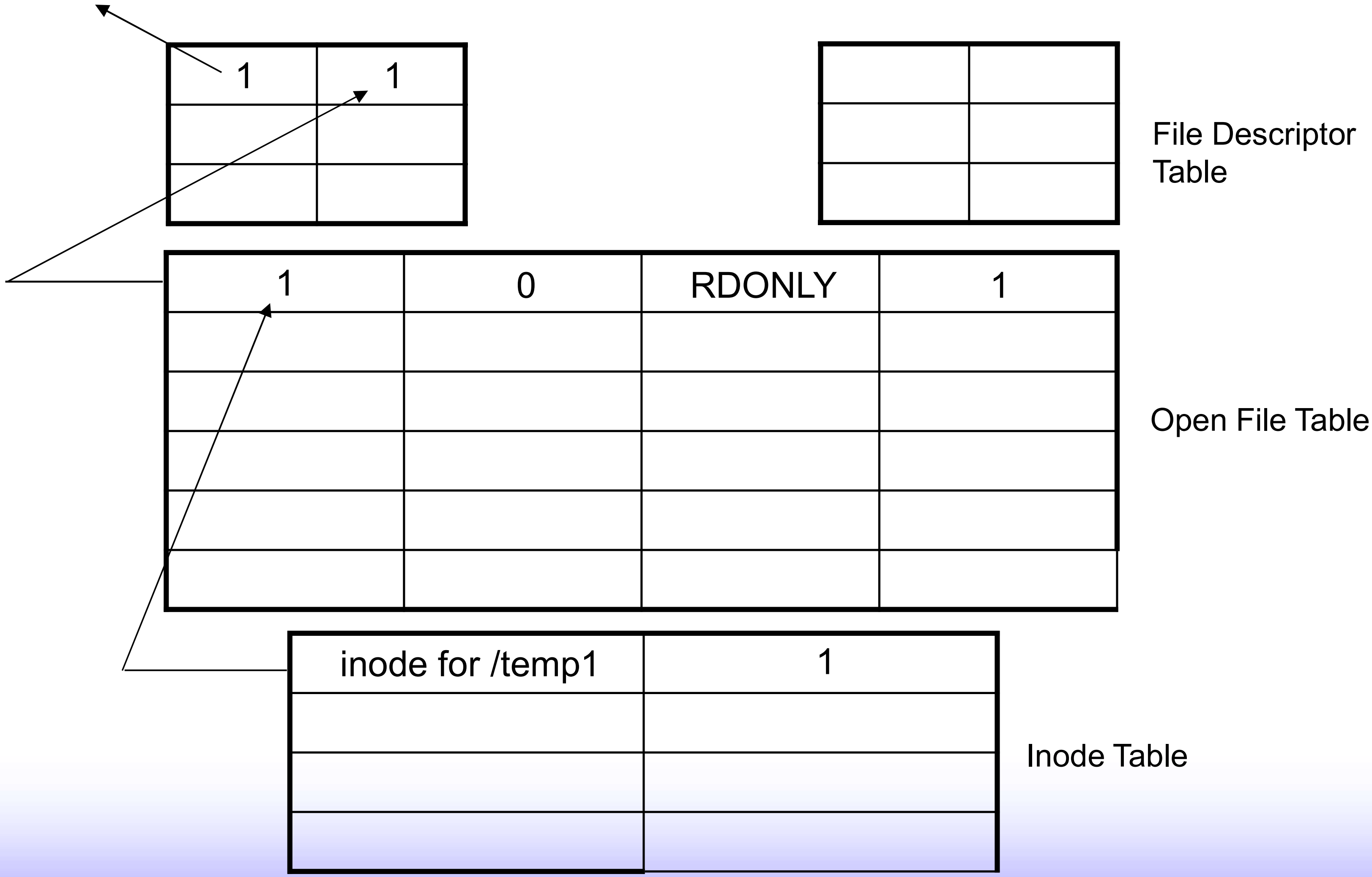
- Note: In response to an "open" call, the OS:
  - Creates an Inode Table entry (including reading the file's inode from disk).
  - Creates a Open File Table entry.
  - Creates a FDT entry (using the next free slot in the FDT table).
- Note: In response to a "close" call, the OS:
  - Clears the FDT entry.
  - Clears the Open File Table entry (unless other FDT entries point to it).
  - Clears Inode Table entry (unless other Open File Table entries point to it).

Open Files - continued

- Example:
  - Process 1: fd1a = open("/temp1", O_RDONLY);
  - Process 2: fd2a = open("/temp1", O_RDWR);
  - Process 1: fd1b = open("/temp2", O_RDWR);
  - Process 2: fd2b = open("/temp2", O_WRONLY);
  - Process 1: fd1c = open("/temp3", O_RDONLY);
  - Process 2: fd2c = open("/temp4", O_WRONLY);

Open Files - continued

- Process 1: fd1a = open("/temp1", O_RDONLY); -- 1



Open Files - continued

- Process 2: fd2a = open("/temp1", O_RDWR); -- 1

1	1

File Descriptor Table

1	2

Open File Table

1	0	RDONLY	1 -> 2
1	0	RDWR	2

Inode Table

inode for /temp1	1 -> 2

Open Files - continued

- Process 1: fd1b = open("/temp2", O_RDWR); -- 2

1	1
2	**3**

1	2

File Descriptor Table

1	0	RDONLY	2
1	0	RDWR	2
2	**0**	**RDWR**	**1**

Open File Table

inode for /temp1	2
inode for /temp2	**1**

Inode Table

Open Files - continued

- Process 2: fd2b = open("/temp2", O_WRONLY); -- 2

1	1
2	3

1	2
2	**4**

File Descriptor Table

1	0	RDONLY	2
1	0	RDWR	2
2	0	RDWR	**1 -> 2**
2	**0**	**WRONLY**	**2**

Open File Table

inode for /temp1	2
inode for /temp2	**1 -> 2**

Inode Table

Open Files - continued

- Process 1: fd1c = open("/temp3", O_RDONLY); -- 3

1	1
2	3
3	**5**

1	2
2	4

File Descriptor Table

1	0	RDONLY	2
1	0	RDWR	2
2	0	RDWR	2
2	0	WRONLY	2
3	**0**	**RDONLY**	**1**

Open File Table

inode for /temp1	2
inode for /temp2	2
inode for /temp3	**1**

Inode Table

Open Files - continued

- Process 2: fd2c = open("/temp4", O_WRONLY); -- 3

1	1
2	3
3	5

1	2
2	4
3	**6**

File Descriptor Table

1	0	RDONLY	2
1	0	RDWR	2
2	0	RDWR	2
2	0	WRONLY	2
3	0	RDONLY	1
4	**0**	**WRONLY**	**1**

Open File Table

inode for /temp1	2
inode for /temp2	2
inode for /temp3	1
inode for /temp4	**1**

Inode Table

Open Files - continued

- Process 2: write(fd2c, buf, 10); -- 3

1	1
2	3
3	5

1	2
2	4
3	**6**

File Descriptor Table

1	0	RDONLY	2
1	0	RDWR	2
2	0	RDWR	2
2	0	WRONLY	2
3	0	RDONLY	1
4	**10**	**WRONLY**	**1**

Open File Table

inode for /temp1	2
inode for /temp2	2
inode for /temp3	1
inode for /temp4	**1**

Inode Table

Open Files - continued

- Process 1: close(fd1b); -- 2

1	1
~~2~~	~~3~~
3	5

1	2
2	4
3	6

File Descriptor Table

1	0	RDONLY	2
1	0	RDWR	2
~~2~~	~~0~~	~~RDWR~~	~~2~~
2	0	WRONLY	**1**
3	0	RDONLY	1
4	10	WRONLY	1

Open File Table

inode for /temp1	2
inode for /temp2	**1**
inode for /temp3	1
inode for /temp4	1

Inode Table

Open Files - continued

- Process 2: close(fd2b); -- 2

1	1
3	5

File Descriptor Table

1	2
~~2~~	~~4~~
3	6

Open File Table

1	0	RDONLY	2
1	0	RDWR	2
~~2~~	~~0~~	~~WRONLY~~	~~1~~
3	0	RDONLY	1
4	10	WRONLY	1

Inode Table

inode for /temp1	2
~~inode for /temp2~~	~~1~~
inode for /temp3	1
inode for /temp4	1

Open Files - continued

File Descriptor Table

1	1
3	5

1	2
3	6

Open File Table

1	0	RDONLY	2
1	0	RDWR	2
3	0	RDONLY	1
4	10	WRONLY	1

Inode Table

inode for /temp1	2
inode for /temp3	1
inode for /temp4	1

Open File Locking

- Provided by some operating systems and file systems
- Mediates access to a file
- Example: system log files

- Types:
  - **Shared lock** – **reader lock**; <u>several</u> processes
  - **Exclusive lock** – **writer lock**; <u>single</u> process

  - **Mandatory** – access is denied depending on locks held and requested.
  - **Advisory** – processes can find status of locks and decide what to do.

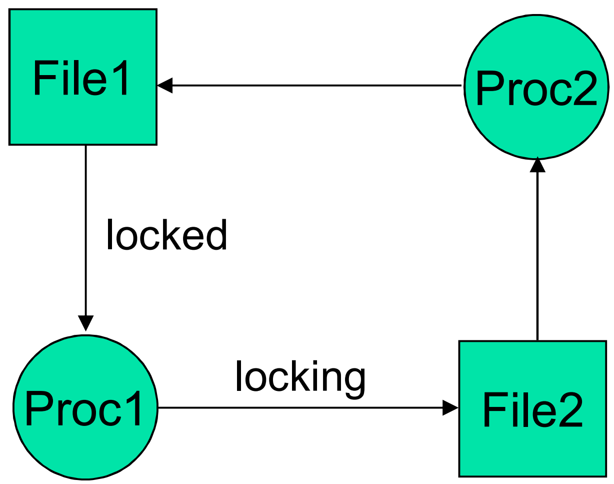
File Locking Example – Java API

```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
   public static final boolean EXCLUSIVE = false;
   public static final boolean SHARED = true;
   public static void main(String arsg[]) throws IOException {
         FileLock sharedLock = null;
         FileLock exclusiveLock = null;
         try {
             RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");
             // get the channel for the file
             FileChannel ch = raf.getChannel();
             // this locks the first half of the file - exclusive
             exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);
             /** Now modify the data . . . */
             // release the lock
             exclusiveLock.release();
```

File Locking Example – Java API

```
            // this locks the second half of the file - shared
            sharedLock = ch.lock(raf.length()/2+1, raf.length(), SHARED);
            /** Now read the data . . . */
            // release the lock
            sharedLock.release();
        } catch (java.io.IOException ioe) {
            System.err.println(ioe);
        } finally {
            if (exclusiveLock != null)
                exclusiveLock.release();
            if (sharedLock != null)
                sharedLock.release();
        }
    }
}
```

File Locking Example – Dead Lock



File Types – Name, Extension

- See Figure 10.2.

2. Access Methods

- **Sequential access**

  read next block
  write next block
  no read after last write
  (rewrite)

  - ☺ *Example?*
  - See Figure 10.3.
- **Direct access or relative access**

  read *n*
  write *n*
  position to *n*
  read next
  write next
  rewrite *n*

  n = relative **block number**
  fixed sized record or block
  ☺ *Example?*

Other Access Methods – Example of Index and Relative Files

- See Figure 10.5.
  - Index file is in main memory.
  - Relative file is on hard disk.
  - ☺ *What is the advantage?*

3. Directory Structure

- Partitions, minidisks, volumes
  - Partitioning is a means to divide a single hard drive into many logical drives.
  - A partition is a contiguous set of blocks on a drive that are treated as an independent disk.
  - A **partition table** is an index that relates sections of the hard drive to partitions.
  - Typically, at least, one partition is on a disk.
  - Some systems allow partitions to be larger than a disk, so that disks can be grouped into one logical units.

- ☺ *Why have multiple partitions?*
  - Encapsulate your data.
    - Since file system corruption is local to a partition, you stand to lose only some of your data if an accident occurs.
  - Increase disk access speed.
    - The **super block** contains a description of the basic size and shape of this file system. The information within it allows the file system manager to use and maintain the file system.
    - Usually only the super block is read when the file system is **mounted**.
    - Small super blocks allow fast access to files.

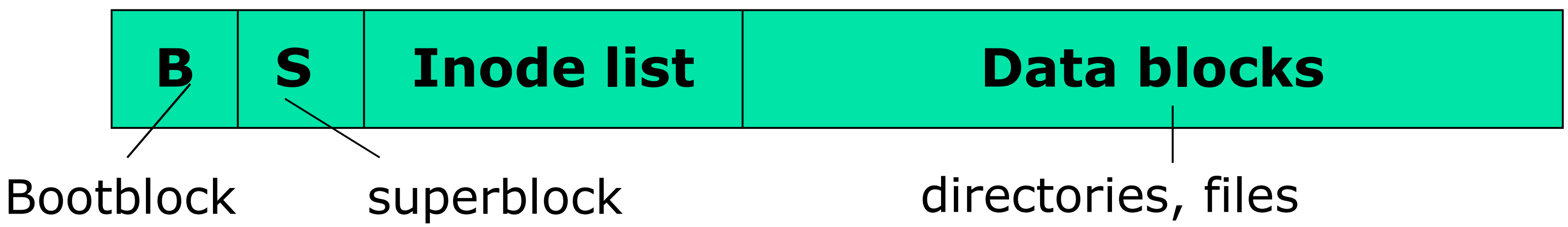
- ☺ *Why have multiple partitions?* - continued
  - Increase disk space efficiency.
    - You can format partitions with varying block sizes, depending on your usage. If your data is in a large number of small files (less than 1k) and your partition uses 4k sized blocks, you are wasting 3k for every file. In general, you waste on average one half of a block for every file, so matching block size to the average size of your files is important if you have many files.
  - Limit data growth.
    - Runaway processes or maniacal users can consume so much disk space that the operating system no longer has room on the hard drive for its bookkeeping operations. This will lead to disaster. By segregating space, you ensure that things other than the operating system die when allocated disk space is exhausted.
  - Different structure for a different partition is possible on the same system.
    - Multiple operating systems

- Example: DOS-type partition tables
  - **Partition table sector**
    - Sector 0 of the disk
    - Called **MBR (Master Boot Record)**
    - Four **partition descriptors** of 16 bytes from offset 446
      - 0 Boot indicator
      - 1-3 Begin CHS (Cylinder/Head/Sector)
      - 4 Partition type
        - 05, 0f, 85 (hex) – **extended partition** for DOS, Window 95, Linux respectively
        - Others – **primary partition**
      - 5-7 End CHS
      - 8-11 Partition start
      - 12-15 Partition size
        - 0 – unused

      - E.g., three primary partitions and one extended partition
      - Windows can boot only from a primary partition.

- Extended partition
  - The first sector is used as a partition table sector again.
  - => **logical partitions**, or inner extended partition

- Primary or logical partition

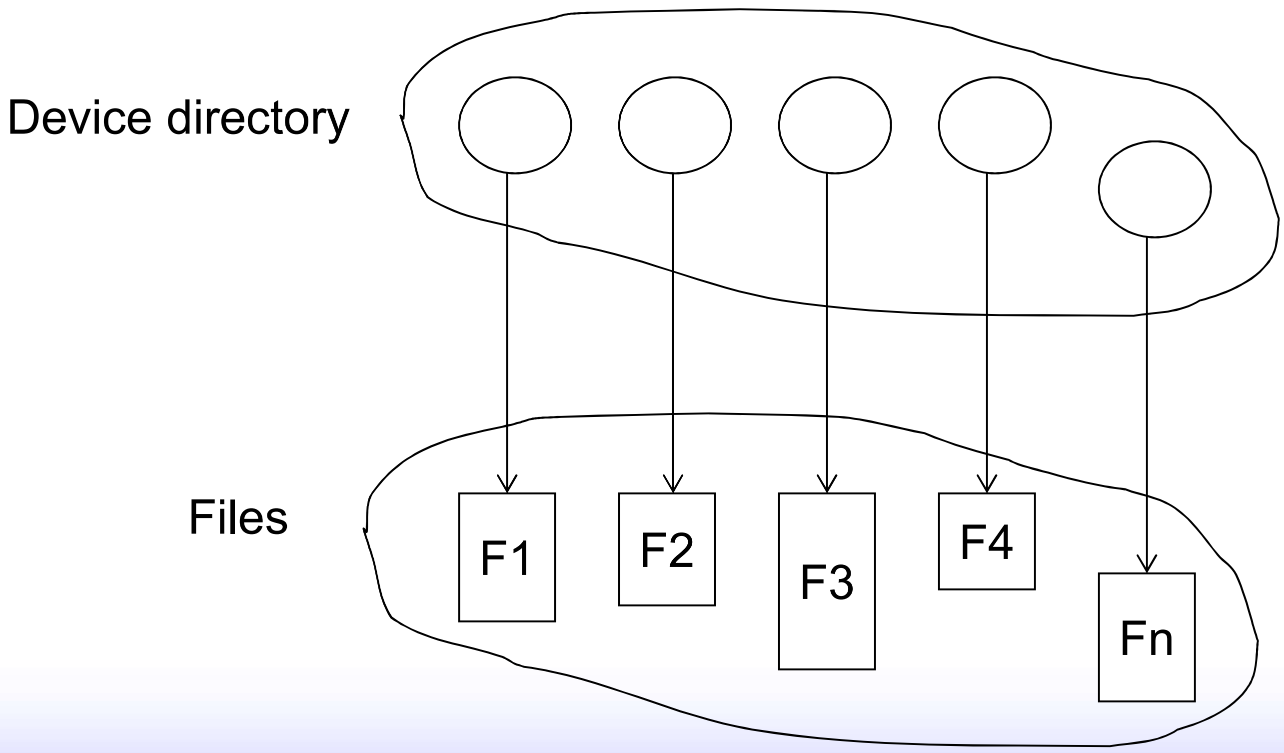


  - Example of UNIX
  - Could be different from different file systems

- Reference
  - Minimal partition table specification – http://www.win.tue.nl/~aeb/partitions/partition_tables.html

Information in a Directory

- **Device directory** (simply **directory**)
  - Information about files in the partition



Information in a Directory – cont.

- For each file:
  - Name
  - Type
  - Address
  - Current length
  - Maximum length
  - Date last accessed (for archival)
  - Date last updated (for dump)
  - Owner ID
  - Protection information (discuss later)

```
[mlee@cs mlee]$ ls -l Tmp
total 716660
-rwxr-xr-x    1 mlee     user           751 Jan 28 13:19 check
-rw-rw-r--    1 mlee     user           513 Feb  3 09:51 First.class
-rw-rw-r--    1 mlee     user           443 Feb  3 08:52 First.java
```

Operations Performed on Directory

- Search for a file.
- Create a file.
- Delete a file.
- List a directory.
- Rename a file.
- Traverse the file system.

Organize the Directory (Logically) to Obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files.
  - The same file can have several different names.
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

- A single directory for all users
- See Figure 10.8.
- ☺ *Problems?*
  - Naming
  - Grouping

Two-Level Directory

- Separate directory for each user
- See Figure 10.9.
- Additional syntax is needed.
  - Path name – e.g., /user3/test
- Relative advantages
  - Can have the same file name for different user
- Still shortcomings
  - Naming
  - Grouping

Tree-Structured Directories

- See Figure 10.10.

Tree-Structured Directories – cont.

- Efficient searching
- Grouping Capability
- **Current directory** (working directory)
  - $ *cd* /spell/mail/prog
  - $ *cat* list

Tree-Structured Directories – cont.

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

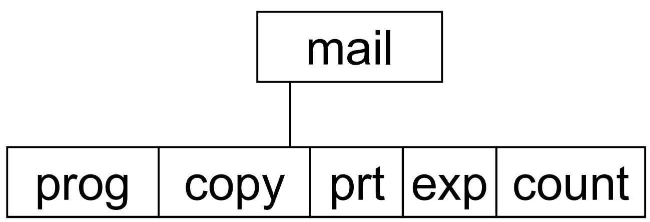
  rm <file-name>

- Creating a new subdirectory is done in current directory

  mkdir <dir-name>

  Example: if in current directory /programs/mail

  mkdir count



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

Acyclic-Graph Directories

- Have shared subdirectories and files
- See Figure 10.11.

Acyclic-Graph Directories – cont.

- Two different names (aliasing)
- If *dict* deletes *count*, delete it or not
- Delete => Dangling pointer
  - Solutions:
    - Backpointers, so we can delete all pointers
      Variable size records a problem
    - Backpointers using a daisy chain organization
    - Entry-hold-count solution

General Graph Directory

- See Figure 10.12.

General Graph Directory - continued

- ☺ *How do we guarantee no cycles?*
  - Allow only links to file not subdirectories
  - Garbage collection
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK.

General Graph Directory - continued

- **$ ln -s . mh**

```
[mlee@cs mh]$ cd mh
[mlee@cs mh]$ pwd
/home2/mlee/mh/mh
[mlee@cs mh]$ ls
Backup - Dec. 21, 2004  Folder Settings  MTAII        Sam Park's English
BackupOfLiver           mail             MTI          Tmp
bin                     mh               public_html  typescript
[mlee@cs mh]$ ls -l
total 44
drwxrwxr-x    3 mlee     user         4096 Dec 21 08:43 Backup - Dec. 21, 2004
drwxrwx---   20 mlee     user         4096 Dec 17 16:33 BackupOfLiver
drwxrwxr-x    3 mlee     user         4096 Feb  3 08:51 bin
drwxrwx---    2 mlee     user         4096 Dec 17 16:33 Folder Settings
drwxrwx---    2 mlee     user         4096 Dec 17 16:34 mail
lrwxrwxrwx    1 mlee     user            1 Feb  4 14:29 mh -> .
drwxrwx---    4 mlee     user         4096 Dec 17 16:33 MTAII
[mlee@cs mh]$
[mlee@cs mh]$ cd mh
[mlee@cs mh]$ pwd
/home2/mlee/mh/mh
[mlee@cs mh]$
```

4. File System Mounting

- A file system must be **mounted** before it can be accessed.
- A unmounted file system is mounted at a **mount point.**
- See Figure 10.13 – 10.14.

5. File Sharing

- Sharing of files on multi-user systems is desirable.
- Sharing may be done through a **protection** scheme.
- On distributed systems, files may be shared across a network.
- Network File System (NFS) is a common distributed file-sharing method.
  - NFS (Network File System)
  - SMB (Server Message Block) / CIFS (Common Internet File System)
  - AFS (Andrew File System)
  - …

File Sharing – Multiple Users

- **User IDs** identify users, allowing permissions and protections to be per-user.
- **Group IDs** allow users to be in groups, permitting group access rights.

6. Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom

- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

```
[mlee@cs mlee]$ ls -l Tmp
total 716660
-rwxr-xr-x    1 mlee     user           751 Jan 28 13:19 check
-rw-rw-r--    1 mlee     user           513 Feb  3 09:51 First.class
-rw-rw-r--    1 mlee     user           443 Feb  3 08:52 First.java
```

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users:

			RWX
a) **owner access**	7	$\Rightarrow$	1 1 1
			RWX
b) **group access**	6	$\Rightarrow$	1 1 0
			RWX
c) **public access**	1	$\Rightarrow$	0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

owner group public

chmod 761 game

Attach a group to a file

chgrp G game