File Systems
A file system is a clearly-defined method that the computer's operating system uses to store, catalog, and retrieve files.
Module 11: File-System Interface

- File Concept
- Access Methods
- Directory Structure
- Protection
- Consistency Semantics
Files & File Systems

• File
  • data, in some format

• File System
  • Set of named files, maybe organized (directories)
  • Information on files (metadata)
Files & File Systems

- alain
- ambients
- assigning_types_to_processes.ps
- asyncrony96.ps.gz
- attelm.ps
- beos.rapport.ps
- BibDeskDocs
- bigraphs
- bis-proof.ps.gz
- BRICS-NS-05-3.pdf
- caml
- causality_types.ps
File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
File Types

- bib
- asyncrony96.ps.gz
- attelm.ps
- assigning_types_to_processes.ps
- beos.rapport.ps
- bis-proof.ps.gz
- causality_types.ps
- BRICS-NS-05-3.pdf
- caml
## File Types – name, extension

<table>
<thead>
<tr>
<th>File Type</th>
<th>Usual extension</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable</td>
<td>exe, com, bin or none</td>
<td>ready-to-run machine-language program</td>
</tr>
<tr>
<td>Object</td>
<td>obj, o</td>
<td>compiled, machine language, not linked</td>
</tr>
<tr>
<td>Source code</td>
<td>c, p, pas, 177, asm, a</td>
<td>source code in various languages</td>
</tr>
<tr>
<td>Batch</td>
<td>bat, sh</td>
<td>commands to the command interpreter</td>
</tr>
<tr>
<td>Text</td>
<td>txt, doc</td>
<td>textual data documents</td>
</tr>
<tr>
<td>Word processor</td>
<td>wp, tex, rrf, etc.</td>
<td>various word-processor formats</td>
</tr>
<tr>
<td>Library</td>
<td>lib, a</td>
<td>libraries of routines</td>
</tr>
<tr>
<td>Print or view</td>
<td>ps, dvi, gif, pdf</td>
<td>ASCII or binary file</td>
</tr>
<tr>
<td>Archive</td>
<td>arc, zip, tar, gz</td>
<td>related files grouped into one file, sometimes compressed.</td>
</tr>
</tbody>
</table>
File Types

(a) An executable file  (b) An archive
File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters.
- Who decides:
  - Operating system
  - Program
File Metadata
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, executing.
- **Time, date, and user identification** – data for protection, security, and usage monitoring.
- Information about files are kept in the directory structure, which is maintained on the disk.
• **Name** – only important for file systems that support long filenames.
• **Type** – needed for some file systems.
• **Location** – important for some applications.
• **Size** – current size of file.
• **Protection** – important for security, and user experience.
• **Time, date, and metadata** – important for some applications.

Information about a file is stored in the file system. Information about a process is maintained on the disk.
File Operations

- create
- write
- read
- reposition within file – file seek
- 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.
Access Methods

- Sequential Access
  - read next
  - write next
  - reset
  - no read after last write
    (rewrite)

- Direct Access
  - read n
  - write n
  - position to n
    - read next
    - write next
  - rewrite n

\( n = \text{relative block number} \)
Sequential-access File
Directories

- alan
- ambients
- assigning_types_to_processes.ps
- asyncrony96.ps.gz
- attelm.ps
- beos.rapport.ps
- BibDeskDocs
- bigraphs
- bis-proof.ps.gz
- BRICS-NS-05-3.pdf
- caml
- causality_types.ps
Directory Structure

- A collection of nodes containing information about all files.

- Both the directory structure and the files reside on disk.
Information in a Device Directory

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID (who pays)
- Protection information (discuss later)
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 Pascal programs, all games, …)
Single-Level Directory

- A single directory for all users.

- Naming problem
- Grouping problem
Two-Level Directory

- Separate directory for each user.

- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability
Tree-Structured Directories
Tree-Structured Directories (Cont.)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - `cd /spell/mail/prog`
  - `type 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 `/spell/mail`

  `mkdir count`

```
    mail
    └── prog
        └── copy
```

- Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”.
Acyclic-Graph Directories

- Have shared subdirectories and files.
Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If $dict$ deletes $list$ $\Rightarrow$ 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.
Using hard links: Time Machine
Using hard links: Time Machine
General Graph Directory

![General Graph Directory Diagram]

- root
  - avi
  - tc
  - jim
- text
  - mail
  - count
  - book
- book
  - mail
  - unhex
  - hyp
- avl
  - count
- unhex
  - hex
General Graph Directory (Cont.)

- 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.
Protection

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

• Types of access
  – Read
  – Write
  – Execute
  – Append
  – Delete
  – List
Access Lists and Groups

- Mode of access: read, write, execute

- Three classes of users
  
a) owner access 7 ⇒ 111
    RWX
  
b) groups access 6 ⇒ 110
    RWX
  
c) public access 1 ⇒ 001
    RWX

- 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.
  
  chmod 761 game

- Attach a group to a file

  chgrp G game
• File-System Structure
• Allocation Methods
• Free-Space Management
• Directory Implementation
• Efficiency and Performance
• Recovery
File-System Structure

• File structure
  – Logical storage unit
  – Collection of related information
• File system resides on secondary storage (disks).
• File system organized into layers.
• *File control block* – storage structure consisting of information about a file.
Typical File-System Organization

Partition A
- Directory
- Files

Partition B
- Directory
- Files

Disk 1

Partition C
- Directory
- Files

Disk 2

Disk 3
File System Implementation

A possible file system layout
Allocation
Putting Bytes on Disk

- File *viewed* as a contiguous sequence of bytes
- Allocation is actually *storing* the bytes
Fragmentation Types

- Data: file not contiguous
- External: unusable empty space between files
- Internal: allocated but unused space
  → file smaller than block
Random Access

- Access time independent of the current block
- Also called Direct Access
- RAM: Random Access Memory
- Tape: no direct access
Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk.
- Simple – only starting location (block #) and length (number of blocks) are required.
- Random access.
- Wasteful of space (dynamic storage-allocation problem).
- Files cannot grow.
Contiguous Allocation of Disk Space
Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.

\[
\text{block} = \begin{array}{c}
\text{pointer}
\end{array}
\]
- Allocate as needed, link together; e.g., file starts at block 9
Linked Allocation (Cont.)

- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Clusters of blocks
  → for better performance (disk head moving)
  → to have fewer pointers

- *File-allocation table (FAT)*: disk-space allocation used by MS-DOS and OS/2.
  The table is a list of entries that maps each cluster number to:
  - the cluster number of the next entry, or
  - an indication this is the last cluster (end of file), or
  - a special entry to mark bad clusters, or
  - a 0 to mark the cluster is unused
  (some cluster may be reserved and are marked in the FAT)
Indexed Allocation

- Brings all pointers together into the *index block*.
- Logical view.

index table
Example of Indexed Allocation

![Diagram of Indexed Allocation]

- **Directory**: Contains a list of files and their corresponding index blocks.
- **File**: "jeep" with index block 19.
- **Index Block**: Contains pointers to the file blocks:
  - 9
  - 16
  - 1
  - 10
  - 25
  - -1
  - -1
  - -1

The diagram illustrates the relationship between the directory, file, and index block in an indexed allocation scheme.
Indexed Allocation (Cont.)

- Need index table
- Random access
- Random access without external fragmentation, but have overhead of index block.
Indexed Allocation – Mapping (Cont.)

outer-index

index table

file
Combined Scheme: UNIX (4K bytes per block)
## Comparing Allocation

<table>
<thead>
<tr>
<th></th>
<th>Random Access</th>
<th>No Data Frag</th>
<th>No External Frag</th>
<th>Space Waste</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contiguous</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>0</td>
</tr>
<tr>
<td>Linked</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td># clusters</td>
</tr>
<tr>
<td>Indexed</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>&gt; # clusters</td>
</tr>
</tbody>
</table>
Free-Space Management

• Bit vector (n blocks)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>n-1</th>
</tr>
</thead>
</table>

\[
\text{bit}[i] = \begin{cases} 
1 & \Rightarrow \text{block}[i] \text{ free} \\
0 & \Rightarrow \text{block}[i] \text{ occupied}
\end{cases}
\]

• Block number calculation

(number of bits per word) * (number of 0-value words) + offset of first 1 bit
Free-Space Management (Cont.)

- Bit map requires extra space. Example:
  
  block size = $2^{12}$ bytes (4K bytes)
  
  disk size = $2^{30}$ bytes (1 gigabyte)
  
  $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

- Easy to get contiguous files

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space

- Grouping

- Counting
Linked Free-Space List on Disk