File Systems

Long-term Information Storage

1. Must store large amounts of data
2. Information stored must survive the termination of the process using it
3. Multiple processes must be able to access the information concurrently

File Naming

<table>
<thead>
<tr>
<th>Extension</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.abc</td>
<td>C source program</td>
</tr>
<tr>
<td>file.gif</td>
<td>Compaqrom Graphical Interchange Format Image</td>
</tr>
<tr>
<td>file.h</td>
<td>Help file</td>
</tr>
<tr>
<td>file.html</td>
<td>World Wide Web HyperText Markup Language document</td>
</tr>
<tr>
<td>file.jpeg</td>
<td>JPEG picture encoded with the JPEG standard</td>
</tr>
<tr>
<td>file.mpg</td>
<td>Movie encoded with the MPEG standard</td>
</tr>
<tr>
<td>file.o</td>
<td>Object file (compiler output, not yet linked)</td>
</tr>
<tr>
<td>file.pdf</td>
<td>Portable Document Format file</td>
</tr>
<tr>
<td>file.ps</td>
<td>PostScript file</td>
</tr>
<tr>
<td>file.tar</td>
<td>Input for the TEA formatting program</td>
</tr>
<tr>
<td>file.txt</td>
<td>General text file</td>
</tr>
<tr>
<td>file.zip</td>
<td>Compressed archive</td>
</tr>
</tbody>
</table>

Typical file extensions.

File Structure

- Three kinds of files
  - byte sequence
  - record sequence
  - tree

File Types

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was mag tape

- Random access
  - bytes/records read in any order
  - essential for data base systems
  - read can be ...
    - move file marker (seek), then read or ...
    - read and then move file marker

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protection</td>
<td>Who can access the file and in what way</td>
</tr>
<tr>
<td>Password</td>
<td>Password needed to access the file</td>
</tr>
<tr>
<td>Creator</td>
<td>ID of the person who created the file</td>
</tr>
<tr>
<td>Owner</td>
<td>Current owner</td>
</tr>
<tr>
<td>Read-only flag</td>
<td>0 for read, 1 for read-only</td>
</tr>
<tr>
<td>Hidden flag</td>
<td>0 for normal, 1 for not display in listings</td>
</tr>
<tr>
<td>System flag</td>
<td>0 for normal, 1 for system file</td>
</tr>
<tr>
<td>Archive flag</td>
<td>0 for archive, 1 for archive file</td>
</tr>
<tr>
<td>Random access flag</td>
<td>0 for sequential access, 1 for random access</td>
</tr>
<tr>
<td>Temporary flag</td>
<td>0 for normal, 1 for delete file on process exit</td>
</tr>
<tr>
<td>Loose flag</td>
<td>0 for unblocked, nonremovable</td>
</tr>
<tr>
<td>Record length</td>
<td>Number of bytes in a record</td>
</tr>
<tr>
<td>Key position</td>
<td>Offset of the key within each record</td>
</tr>
<tr>
<td>Key length</td>
<td>Number of bytes in the key</td>
</tr>
<tr>
<td>Creation time</td>
<td>Date and time the file was created</td>
</tr>
<tr>
<td>Time of last access</td>
<td>Date and time the file was last accessed</td>
</tr>
<tr>
<td>Time of last change</td>
<td>Date and time the file has last changed</td>
</tr>
<tr>
<td>Current size</td>
<td>Number of bytes in the file</td>
</tr>
<tr>
<td>Maximum size</td>
<td>Number of bytes the file may grow to</td>
</tr>
</tbody>
</table>

Possible file attributes

An Example Program Using File System Calls (1/2)

```c
#include <sys/types.h>         /* include necessary header files */
#include <fcntl.h>
#include <unistd.h>

int main(argc, argv[]); /* ANSI prototype */
#define O_RDONLY 0
#define O_RDWR 1
#define O_WRONLY 2
#define O_CREAT 0
#define O_TRUNC 1
#define O_APPEND 2
#define O_EXCL 4

int main(argc, argv[]); { // use a buffer size of 4096 bytes
    char buffer[4096]; // protection bits for output file
    int in_fd, out_fd, rd_count, wt_count;
    if (argc != 3) exit(1); // syntax error if argc is not 3
    } // Close the files
    if (rd_count < 0) break; // if end of file or error, exit loop
    if (wt_count >= 0) exit(4); // if end of file or error, exit loop
    if (rd_count < 0) exit(5); // error on last read
    exit(0); // error on last read
```

Memory-Mapped Files

(a) Segmented process before mapping files into its address space
(b) Process after mapping existing file abc into one segment creating new segment for xyz

Directories: Single-Level Directory Systems

- A single level directory system
  - contains 4 files
  - owned by 3 different people, A, B, and C
Two-level Directory Systems

Letters indicate owners of the directories and files

Hierarchical Directory Systems

A hierarchical directory system

Path Names

A UNIX directory tree

Directory Operations

1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. Link
8. Unlink

File System Implementation

A possible file system layout

Implementing Files (1)

(a) Contiguous allocation of disk space for 7 files
(b) State of the disk after files D and E have been removed
Implementing Files (2)

Storing a file as a linked list of disk blocks

Implementing Files (3)

Linked list allocation using a file allocation table in RAM

Implementing Files (4)

An example i-node

Implementing Directories (1)

(a) A simple directory
fixed size entries
disk addresses and attributes in directory entry
(b) Directory in which each entry just refers to an i-node

Implementing Directories (2)

Two ways of handling long file names in directory
(a) In-line
(b) In a heap

Shared Files (1)

File system containing a shared file
Shared Files (2)

(a) Situation prior to linking
(b) After the link is created
(c) After the original owner removes the file

Disk Structure

Disk Space Management (1)

- Dark line (left hand scale) gives data rate of a disk
- Dotted line (right hand scale) gives disk space efficiency
- All files 2KB

Disk Space Management (2)

(a) Storing the free list on a linked list
(b) A bit map

Disk Space Management (3)

1. Almost-full block of pointers to free disk blocks in RAM
   - three blocks of pointers on disk
2. Result of freeing a 3-block file
3. Alternative strategy for handling 3 free blocks
   - shaded entries are pointers to free disk blocks

Disk Space Management (4)

Quotas for keeping track of each user's disk use
File System Performance (1)

The block cache data structures

File System Performance (2)

- I-nodes placed at the start of the disk
- Disk divided into cylinder groups
  - each with its own blocks and i-nodes