**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.wav</td>
<td>Backup file</td>
</tr>
<tr>
<td>file.c</td>
<td>C source program</td>
</tr>
<tr>
<td>file.gif</td>
<td>CompuServe 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.jpg</td>
<td>Still picture encoded with the JPEG standard</td>
</tr>
<tr>
<td>filempeg</td>
<td>Movie encoded with the MPEG standard</td>
</tr>
<tr>
<td>file.ppt</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.tex</td>
<td>Input for the TEX 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 rewinding or backward
  - convenient when medium was magnetic 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 of the file</td>
</tr>
<tr>
<td>Read-only flag</td>
<td>0 for read/write, 1 for read-only</td>
</tr>
<tr>
<td>Hidden flag</td>
<td>0 for normal, 1 for not display in lllistings</td>
</tr>
<tr>
<td>System flag</td>
<td>0 for normal file, 1 for system file</td>
</tr>
<tr>
<td>Archive flag</td>
<td>0 for has been backed up, 1 for needs to be backed up</td>
</tr>
<tr>
<td>ASCII binary flag</td>
<td>0 for ASCII file, 1 for binary file</td>
</tr>
<tr>
<td>Random access flag</td>
<td>0 for sequential access only, 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>Locked flag</td>
<td>0 for unlocked, non-read for backed</td>
</tr>
<tr>
<td>Record length</td>
<td>Number of bytes in the file</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 field</td>
</tr>
<tr>
<td>Key field</td>
<td>Date and time the file was created</td>
</tr>
<tr>
<td>Date 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 was 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
/* File copy program. Error checking and reporting is minimal. */
#include <sys/types.h> /* include necessary header files */
#include <stdio.h>
#include <unistd.h>

int main(argc, argv[]); /* ANES prototype */
define OUTPUT_MODE 0700
edefine BUFFER_SIZE 4096

main(argc, argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUFFER_SIZE];
    in_fd = open(argv[1], O_RDONLY);
    if (in_fd < 0) exit(2);
    out_fd = creat(argv[2], OUTPUT_MODE);
    if (out_fd < 0) exit(2);
    if (!open_file(out_fd)) {
        printf("failed to create the destination file
" "(out_file) or open file failed
")
        exit(4);
    }
    rd_count = read(in_fd, buffer, BUFFER_SIZE);
    if (rd_count == 0) break;
    if (rd_count == 0) exit(2);
    wt_count = write(out_fd, buffer, rd_count);
    if (wt_count < 0) exit(4);
    if (wt_count == 0) exit(2);
    else exit(5);
    close(in_fd);
    close(out_fd);
    if (rd_count == 0)
        exit(2);
    if (rd_count == 0)
        exit(0);
}``

---

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

---

**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 are located near the start of the disk
- Disk divided into cylinder groups, each with its own blocks and i-nodes

File System API Calls in Windows 2000 (1)

<table>
<thead>
<tr>
<th>Win32 API function</th>
<th>UNIX</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateFile</td>
<td>open</td>
<td>Create a file or open an existing file, return a handle</td>
</tr>
<tr>
<td>DeleteFile</td>
<td>unlink</td>
<td>Destroy an existing file</td>
</tr>
<tr>
<td>CloseHandle</td>
<td>close</td>
<td>Close a file</td>
</tr>
<tr>
<td>ReadFile</td>
<td>read</td>
<td>Read data from a file</td>
</tr>
<tr>
<td>WriteFile</td>
<td>write</td>
<td>Write data to a file</td>
</tr>
<tr>
<td>SetFilePointer</td>
<td>lseek</td>
<td>Set the file pointer to a specific place in the file</td>
</tr>
<tr>
<td>GetFileAttributes</td>
<td>stat</td>
<td>Return the file properties</td>
</tr>
<tr>
<td>LockFile</td>
<td>lseek</td>
<td>Lock a region of the file to provide mutual exclusion</td>
</tr>
<tr>
<td>UnlockFile</td>
<td>lseek</td>
<td>Unlock a previously locked region of the file</td>
</tr>
</tbody>
</table>

- Principle Win32 API functions for file I/O
- Second column gives nearest UNIX equivalent

File System API Calls in Windows 2000 (2)

```c
/* Open files for input and output. */
inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
unless = CreateFile("new", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

/* Copy the file. */
do {
  s = ReadFile(inhandle, buffer, BUF_SIZE, &count, NULL);
  if (s & count > 0) WriteFile(unhandle, buffer, count, &count, NULL);
  while (s > 0 & & count > 0);
}

/* Close the files. */
CloseHandle(inhandle);
CloseHandle(unhandle);
```

A program fragment for copying a file using the Windows 2000 API functions

File System Calls in Windows 2000 (3)

<table>
<thead>
<tr>
<th>Win32 API function</th>
<th>UNIX</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDirectory</td>
<td>mkdir</td>
<td>Create a new directory</td>
</tr>
<tr>
<td>RemoveDirectory</td>
<td>rmdir</td>
<td>Remove an empty directory</td>
</tr>
<tr>
<td>FindFirstFile</td>
<td>opendir</td>
<td>Initialize to start reading the entries in a directory</td>
</tr>
<tr>
<td>FindNextFile</td>
<td>readdir</td>
<td>Read the next directory entry</td>
</tr>
<tr>
<td>MoveFile</td>
<td>rename</td>
<td>Move a file from one directory to another</td>
</tr>
<tr>
<td>SetCurrentDirectory</td>
<td>chdir</td>
<td>Change the current working directory</td>
</tr>
</tbody>
</table>

- Principle Win32 API functions for directory management
- Second column gives nearest UNIX equivalent, when one exists

File System Structure (1)

- The NTFS master file table

The NTFS master file table

- File metadata files
- File allocation table
- File attributes
- Volume metadata
- Boot record
- Boot sector
- First sector
File System Structure (2)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard information</td>
<td>Flag bits, timestamps, etc.</td>
</tr>
<tr>
<td>File name</td>
<td>File name in Unicode; may be repeated for MS-DOS name</td>
</tr>
<tr>
<td>Security descriptor</td>
<td>Obsolete. Security information is now in $ExtDSecuresian; Security information may be repeated for MS-DOS name</td>
</tr>
<tr>
<td>Attribute list</td>
<td>Location of additional MFT records, if needed</td>
</tr>
<tr>
<td>Object ID</td>
<td>64-bit file identifier unique to this volume</td>
</tr>
<tr>
<td>Reparse point</td>
<td>Used for mounting and symbolic links</td>
</tr>
<tr>
<td>Volume name</td>
<td>Name of this volume (used only in $Volume)</td>
</tr>
<tr>
<td>Volume information</td>
<td>Volume version (used only in $Volume)</td>
</tr>
<tr>
<td>Index root</td>
<td>Used for directories</td>
</tr>
<tr>
<td>Index allocation</td>
<td>Used for very large directories</td>
</tr>
<tr>
<td>Bitmap</td>
<td>Used for very large directories</td>
</tr>
<tr>
<td>Logged utility stream</td>
<td>Controls logging to log file</td>
</tr>
<tr>
<td>Data</td>
<td>Stream data, may be repeated</td>
</tr>
</tbody>
</table>

The attributes used in MFT records

File System Structure (3)

An MFT record for a three-run, nine-block file

File System Structure (4)

A file that requires three MFT records to store its runs

File System Structure (5)

The MFT record for a small directory.