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.bak</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.hlp</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>file.mp3</td>
<td>Music encoded in MPEG layer 3 audio format</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.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

(a) An executable file  (b) An archive

November 17, 2003  CS 3204: OS, Fall 2003
File Access

- 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
# 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/write; 1 for read only</td>
</tr>
<tr>
<td>Hidden flag</td>
<td>0 for normal; 1 for do not display in listings</td>
</tr>
<tr>
<td>System flag</td>
<td>0 for normal files; 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>Lock flags</td>
<td>0 for unlocked; nonzero for locked</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 field</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

November 17, 2003

CS 3204: OS, Fall 2003
File Operations

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename
An Example Program Using File System Calls (1/2)

/* File copy program. Error checking and reporting is minimal. */

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

int main(int argc, char *argv[]); /* ANSI prototype */

#define BUF_SIZE 4096           /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700        /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);          /* syntax error if argc is not 3 */
/* Open the input file and create the output file */
in_fd = open(argv[1], O_RDONLY);  /* open the source file */
if (in_fd < 0) exit(2);  /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE);  /* create the destination file */
if (out_fd < 0) exit(3);  /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
    rd_count = read(in_fd, buffer, BUF_SIZE);  /* read a block of data */
    if (rd_count <= 0) break;  /* if end of file or error, exit loop */
    wt_count = write(out_fd, buffer, rd_count);  /* write data */
    if (wt_count <= 0) exit(4);  /* wt_count <= 0 is an error */
}

/* Close the files */
close(in_fd);
close(out_fd);
if (rd_count == 0)  /* no error on last read */
    exit(0);
else
    exit(5);  /* error on last read */
(a) Segmented process before mapping files into its address space

(b) Process after mapping existing file \textit{abc} into one segment creating new segment for \textit{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)

File Attributes
- Address of disk block 0
- Address of disk block 1
- Address of disk block 2
- Address of disk block 3
- Address of disk block 4
- Address of disk block 5
- Address of disk block 6
- Address of disk block 7
- Address of block of pointers

Disk block containing additional disk addresses

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

November 17, 2003
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)

Open file table

- Attributes
  - disk addresses
  - User = 8

- Quota pointer

Quota table

- Soft block limit
- Hard block limit
- Current # of blocks
- # Block warnings left
- Soft file limit
- Hard file limit
- Current # of files
- # File warnings left

Quotas for keeping track of each user’s disk use

November 17, 2003
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
### 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>fcntl</td>
<td>Lock a region of the file to provide mutual exclusion</td>
</tr>
<tr>
<td>UnlockFile</td>
<td>fcntl</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**
/* Open files for input and output. */
  inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
  outhandle = CreateFile("newf", 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(outhandle, buffer, count, &ocnt, NULL);
} while (s > 0 && count > 0);

/* Close the files. */
CloseHandle(inhandle);
CloseHandle(outhandle);

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

November 17, 2003
CS 3204: OS, Fall 2003
## 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 $Extend$Secure</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 $LogFile</td>
</tr>
<tr>
<td>Data</td>
<td>Stream data; may be repeated</td>
</tr>
</tbody>
</table>

### The attributes used in MFT records

November 17, 2003

CS 3204: OS, Fall 2003
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)

A directory entry contains the MFT index for the
the length of the file name, the file name itself,
and various fields and flags

The MFT record for a small directory.
Figure 12.13  UNIX Block Addressing Scheme

CS 3204: OS, Fall 2003