Last Class: Memory management

- Page replacement algorithms - make paging work well.
  - Random, FIFO, MIN, LRU
  - Approximations to LRU: Second chance
  - Multiprogramming considerations

Today: File System Functionality

Remember the high-level view of the OS as a translator from the user abstraction to the hardware reality.

User Abstraction		Hardware Resource
Processes/Threads		CPU
Address Space	$\Longleftarrow$ **OS** $\Longrightarrow$	Memory
Files		Disk

File System Abstraction

Applications Daemons Servers Shell

Programmer Interface

Open() Close() Read() Write()
Link() Rename()

Device Independent Interface

Sectors Tracks
Seek() ReadBlock() WriteBlock()

Device Interface

Hardware
Disk

User Requirements on Data

- **Persistence:** data stays around between jobs, power cycles, crashes
- **Speed:** can get to data quickly
- **Size:** can store lots of data
- **Sharing/Protection:** users can share data where appropriate or keep it private when appropriate
- **Ease of Use:** user can easily find, examine, modify, etc. data

Hardware/OS Features

- Hardware provides:
  - **Persistence:** Disks provide non-volatile memory
  - **Speed:** Speed gained through random access
  - **Size:** Disks keep getting bigger (typical disk on a PC=20GB)
- OS provides:
  - **Persistence:** redundancy allows recovery from some additional failures
  - **Sharing/Protection:** Unix provides read, write, execute privileges for files
  - **Ease of Use**
    - Associating names with chunks of data (files)
    - Organize large collections of files into directories
    - Transparent mapping of the user's concept of files and directories onto locations on disks

Files

- **File:** Logical unit of storage on a storage device
  - Formally, named collection of related information recorded on secondary storage
  - **Example:** reader.cc, a.out
- Files can contain programs (source, binary) or data
- Files can be structured or unstructured
  - Unix implements files as a series of bytes (unstructured)
  - IBM mainframes implements files as a series of records or objects (structured)
- File attributes: name, type, location, size, protection, creation time

User Interface to the File System

Common file operations:

Data operations:

- Create()
- Open()
- Read()
- Delete()
- Close()
- Write()
- Seek()

Naming operations:

- HardLink()
- SoftLink()
- Rename()

Attributes (owner, protection,...):

- SetAttribute()
- GetAttribute()

OS File Data Structures

1. Open file table - shared by all processes with an open file.
   - open count
   - file attributes, including ownership, protection information, access times, ...
   - location(s) of file on disk
   - pointers to location(s) of file in memory
2. Per-process file table - for each file,
   - pointer to entry in the open file table
   - current position in file (offset)
   - mode in which the process will access the file (r, w, rw)
   - pointers to file buffer

File Operations: Creating a File

- **Create(name)**
  - Allocate disk space (check disk quotas, permissions, etc.)
  - Create a file descriptor for the file including name, location on disk, and all file attributes.
  - Add the file descriptor to the directory that contains the file.
  - Optional file attribute: file type (Word file, executable, etc.)
    * **Advantages:** better error detection, specialized default operations (double-clicking on a file knows what application to start), enables storage layout optimizations
    * **Disadvantages:** makes the file system and OS more complicated, less flexible for user.
    * Unix opts for simplicity (no file types), Macintosh/Windows opt for user-friendliness

File Operations: Deleting a File

- **Delete(name)**
  - Find the directory containing the file.
  - Free the disk blocks used by the file.
  - Remove the file descriptor from the directory.

File Operations: Opening and Closing Files

- **fileId = Open(name, mode)**
  - Check if the file is already open by another process. If not,
    * Find the file.
    * Copy the file descriptor into the system-wide open file table.
  - Check the protection of the file against the requested mode. If not ok, abort
  - Increment the open count.
  - Create an entry in the process's file table pointing to the entry in the system-wide file table. Initialize the current file pointer to the start of the file.
- **Close(fileId)**
  - Remove the entry for the file in the process's file table.
  - Decrement the open count in the system-wide file table.
  - If the open count == 0, remove the entry in the system-wide file table.

OS File Operations: Reading a File

- **Read(fileID, from, size, bufAddress)** - random access
  - OS reads "size" bytes from file position "from" into "bufAddress"

```
for (i = from; i < from + size; i++)
     bufAddress[i - from] = file[i];
```

- **Read(fileID, size, bufAddress)** - sequential access
  - OS reads "size" bytes from current file position, fp, into "bufAddress" and increments current file position by size

```
for (i = 0; i < size; i++)
     bufAddress[i] = file[fp + i];
fp += size;
```

OS File Operations

- **Write** is similar to reads, but copies from the buffer to the file.
- **Seek** just updates fp.
- **Memory mapping** a file
  - Map a part of the portion virtual address space to a file
  - Read/write to that portion of memory $\Rightarrow$OS reads/writes from corresponding location in the file
  - File accesses are greatly simplified (no read/write call are necessary)

File Access Methods

- Common file access patterns from the programmer's perspective
  - **Sequential:** data processed in order, a byte or record at a time.
    * Most programs use this method
    * *Example:* compiler reading a source file.
  - **Keyed:** address a block based on a key value.
    * *Example:* database search, hash table, dictionary
- Common file access patterns from the OS perspective:
  - **Sequential:** keep a pointer to the next byte in the file. Update the pointer on each read/write.
  - **Random:** address any block in the file directly given its offset within the file.

Naming and Directories

- Need a method of getting back to files that are left on disk.
- OS uses numbers for each files
- Users prefer textual names to refer to files.
  - **Directory:** OS data structure to map names to file descriptors
- Naming strategies
  - **Single-Level Directory:** One name space for the entire disk, every name is unique.
    1. Use a special area of disk to hold the directory.
    2. Directory contains <name, index> pairs.
    3. If one user uses a name, no one else can.
    4. Some early computers used this strategy. Early personal computers also used this strategy because their disks were very small.
  - **Two Level Directory:** each user has a separate directory, but all of of each user's files must still have unique names

Naming Strategies (continued)

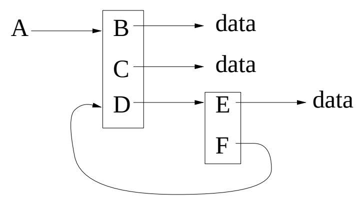
- Multilevel Directories - tree structured name space (Unix, and all other modern operating systems).
  1. Store directories on disk, just like files except the file descriptor for directories has a special flag bit.
  2. User programs read directories just like any other file, but only special system calls can write directories.
  3. Each directory contains <name, fileDesc> pairs in no particular order. The file referred to by a name may be another directory.
  4. There is one special root directory. *Example:* How do we look up name: /usr/local/bin/netscape

Referential naming

- Hard links (Unix: *ln* command)
  - A hard link adds a second connection to a file
  - *Example:* creating a hard link from B to A

Initially:	A → file #100
After "ln A B":	A → file #100
	B → file #100

  - OS maintains reference counts, so it will only delete a file after the last link to it has been deleted.
  - *Problem:* user can create circular links with directories and then the OS can never delete the disk space.
  - *Solution:* No hard links to directories



Referential Naming

- Soft links (Unix: *ln -s* command)
  - A soft link only makes a symbolic pointer from one file to another.
  - *Example:* creating a soft link from B to A

Initially:	A → file #100
After "ln A B":	A → file #100
	B → A

  - removing B does not affect A
  - removing A leaves the name B in the directory, but its contents no longer exists
  - *Problem:* circular links can cause infinite loops (e.g., trying to list all the files in a directory and its subdirectories)
  - *Solution:* limit number of links traversed.

Directory Operations

- Search for a file: locate an entry for a file
- Create a file: add a directory listing
- Delete a file: remove directory listing
- List a directory: list all files (*ls* command in UNIX)
- Rename a file
- Traverse the file system

Protection

- The OS must allow users to control sharing of their files ⇒control access to files
- Grant or deny access to file operations depending on protection information
- **Access lists and groups** (Windows NT)
  - Keep an access list for each file with user name and type of access
  - Lists can become large and tedious to maintain
- **Access control bits** (UNIX)
  - Three categories of users (owner, group, world)
  - Three types of access privileges (read, write, extcute)
  - Maintain a bit for each combination (111101000 = rwxr-x—)

Summary of File System Functionality

- Naming
- Protection
- Persistence
- Fast access