abstraction to the hardware reality.

Remember the high-level view of the OS as a translator from the user

---

## Today: File System Functionality

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

---

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

**User Requirements on Data**

---

**File System Abstraction**

```
+--------------------------+
| DISK Hardware            |
| +------------------------+           +------------------------+           +------------------------+
| | Device Interface       |           | Device Interface       |           | Device Interface       |
| |                        |           |                        |           |                        |
| | Sections               |           | Tracks                 |           | L() Rename()           |
| |                        |           |                        |           |                        |
| | Read() Write()         |           |                        |           |                        |
| |                        |           |                        |           |                        |
| |                        |           |                        |           | Open() Close() Read() Write() |
| |                        |           |                        |           |                        |
| |                        |           |                        |           |                        |
| +------------------------+           +------------------------+           +------------------------+
| Applications Daemons Servers Shell |
| +--------------------------+
```
File attributes: name, type, location, size, protection, creation time

- IBM mainframe implements files as a series of records or objects (structured)
- Unix implements files as a series of bytes (unstructured)
- Files can be structured or unstructured
- Files can contain programs (source, binary) or data

Example: Reader.cc: a.out
- Formally: named collection of related information recorded on secondary storage
- File: Logical unit of storage on a storage device

**Files**

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

**Hardware/OS Features**
OS File Structures

1. Open file table - shared by all processes with an open file.

2. Per-process file table - for each file:

   - Pointers to the buffer
   - Mode in which the process will access the file (r, w, rw)
   - Current position in the file (offset)
   - Pointer to entry in the open file table

Naming operations:

- Rename()
- Getattribute()
- Softlink()
- Setattribute()
- Hardlink()

Attributes (owner, protection...):

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

Data operations:

Common file operations:

User interface to the file system
- Remove the file descriptor from the directory.
- Free the disk blocks used by the file.
- Find the directory containing the file.

Delete(name)

File Operations: Deleting a File

 advantages:
- Optional: add attributes to the file descriptor (name, location on disk, etc.).
- Create a file descriptor for the file (including name, location on disk, etc.).
- Allocate disk space (check disk quotas, permissions, etc.).

user-friendliness

Disk operations: makes the file system and OS more complicated, less flexible for

Unix: opt for simplicity (no file types), Macintosh/Windows opt for

user.

Disadvantages: makes the file system and OS more complicated, less flexible for

on a file knows what application to start, enables storage/layering for

advantages: better error detection, specialized default operations (double-clicking)

Create(name)

File Operations: Creating a File

Department of Computer Science, Utah State University
Read(field, size, buffer address) - sequential access

Read(field, from, size, buffer address) - random access

OS File Operations: Reading a File

Close(field)

Field Operations: Opening and Closing Files

If the open count == 0, remove the entry in the system-wide file table.
- Decrement the open count in the system-wide file table.
- Remove the entry for the file in the process's file table.

If the open count == 1, copy the file descriptor into the system-wide open file table.
- Check the protection of the file to ensure the requested mode. If not ok, abort.
- Check if the file is already open by another process. If not,

Field = Open(name, mode)
- **Random**: address any block in the file directly given its offset within the file.
- **Sequential**: keep a pointer to the next byte in the file. Update the pointer on each read/write.

**Common file access patterns from the OS perspective**:

- Example: database search, hash table, dictionary
- **Keyed**: address a block based on a key value
  - Example: complete reading a source file
    - Most programs use this method
- **Sequential**: data processed in order, a byte or record at a time

**File Access Methods**

- **Memory mapping a file**
  - File accesses are greatly simplified (no read/write call are necessary)
  - Location in the file
  - Read/write to that portion of memory \(\leftrightarrow\) OS reads/writes from corresponding
  - Map a part of the portion virtual address space to a file

**OS File Operations**

- **Seek just updates fp**.
- **Write is similar to reads**, but copies from the buffer to the file.
%usr/local/bin/escape

4. There is one special root directory. Example: How do we look up name?

refers to a name may be another directory. The file

3. Each directory contains <name, index> pairs in no particular order. The file
directories just like any other file. But only special system calls

2. Special file

1. Store directories on disk. Just like files except the file descriptor for directories has a

* modern operating systems)

Multi-level Directories – Free Structure: Name space (union) and all other

| Naming Strategies (continued) |

| Naming Strategies |

- Two-level directory: Each user has a separate directory, but all of each user’s

structures because their disks were very small.

- Single-level directory: One name space for the entire disk. Every name is unique.

- Directory: OS data structure to map names to the descriptors.

- Users prefer textual names to refer to files.

- OS uses numbers for each file.

Need a method of getting back to files that are left on disk.

| Naming and Directories |
Solution: limit number of links traversed.

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

\[
\begin{align*}
B & \rightarrow A \\
\text{After } \text{in } A \text{ B:} & \quad A \rightarrow \#100 \\
\text{Initially:} & \quad A \rightarrow \#100
\end{align*}
\]

Example: creating a soft link from B to A
- A soft link only makes a symbolic pointer from one file to another.

Soft links (unix: \texttt{ln -s command})

Referential Naming

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

\[
\begin{align*}
B & \rightarrow \#100 \\
\text{After } \text{in } A \text{ B:} & \quad A \rightarrow \#100 \\
\text{Initially:} & \quad A \rightarrow \#100
\end{align*}
\]

Example: creating a hard link from B to A
- A hard link adds a second connection to a file

Hard links (unix: \texttt{ln command})

Referential Naming