Abstract 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
User Requirements on Data

- 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
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

Disks

- Transparent mapping of the user’s concept of files and directories onto locations on
  transparent storage device
- Organizes large collections of files into directories
- Associating names with 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 provides:

Features
OS File Data Structures

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

2. Per-process file table - for each file:
   - pointers to file buffer
   - mode in which the process will access the file (r, w, rw)
   - current position in file (offset)
   - pointer to entry in the open file table

- Rename()
- Getattribute()
- Openlink()
- Setattribute()
- Hardlink()

Naming operations: attributes (owner, protection,.

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

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

Create(name)

File Operations: Creating a File
1. If the open count == 0, remove the entry in the system-wide file table.
2. Decrement the open count in the system-wide file table.
3. Remove the entry for the file in the process's file table.

Close(file)

- Create an entry in the process's file table pointing to the entry in the system-wide file table.
- Increment the open count.
- Check the protection of the file against the requested mode. If not ok, abort.
- Check if the file is already open by another process. If not,
  
  - Field = Open(name, mode)

OS File Operations: Reading a File

- OS reads `size` bytes from current file position, jumps into `bufferaddress` and sequential access
- `Read(file, size, bufferaddress)`

- OS reads `size` bytes from current file position, jumps into `bufferaddress` and random access
- `Read(file, from, size, bufferaddress)`

File Operations: Opening and Closing Files

- `open`
File Access Methods

Example: database search, hash table, dictionary
Keyed: access a block based on a key value
Example: compile reading a source file
Most programs use this method
Sequential: data processed in order, a byte at a time

Common file access patterns from the programmer's perspective

File Operations

- Random: address any block in the file directly given its offset within the file
- Read/Write:
  - Sequential: keep a pointer to the next byte in the file. Update the pointer on each block.

Common file access patterns from the OS perspective

- Map a part of the portion virtual address space to a file
- Memory mapping a file
- Seek just updates fp
- Write is similar to reads, but copies from the buffer to the file
There is a directory for each of the users. The root directory is used to organize files. Each user has their own directory. Here are two methods:

- **Single-level Directory:**
  - Each directory has a single level of files.
  - Files must have unique names.

- **Two-level Directory:**
  - Each user has a separate directory.
  - Each directory contains a file called `.index`.
  - Each file has a unique name.

OS systems tend to use the second method, as it requires fewer disk accesses.
Referential Naming

- Solution: No hard links to directory
  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

\[
\begin{align*}
B & \rightarrow \text{file} \\
A & \rightarrow \text{file} \\
\end{align*}
\]

Initially

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

Referential Naming

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

\[
\begin{align*}
B & \rightarrow A \\
\text{After `ln A B`} & : A \rightarrow \text{file} \\
\text{Initially} & : A \rightarrow \text{file} \\
\end{align*}
\]
- Maintain a bit for each combination (11110100 = read-x--)
- Three types of access privileges (read, write, execute)
- Three categories of users (owner, group, world)

**Access control bits (UNIX)**

- Lists can become large and tedious to maintain
- Keep an access list for each file with user name and type of access

**Access lists and Groups (Windows NT)**

- Grant or deny access to file operations depending on protection
- To files
- The OS must allow users to control sharing of their files = control access

**Protection**

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

**Directory Operations**
Summary of File System Functionality

- Naming
- Protection
- Persistence
- Fast access