Disk  | Memory  | OS  | Files  | Address Space  | Process/Threads  | User Abstraction
Memory  | CPU  | Hardware Resource  |

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

Today: File System Functionality

- Multiprogramming considerations
- Approximation 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
- **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**

**Disk Hardware**

- Device Interface
- Independent Device Interface
- Programmer Interface
- Applications/Daemons/Servers
- Shell
- Tracks
- Sectors
- ReadBlock(), WriteBlock()
- Seek()

**File System Abstraction**
- 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
  - Organize large collections of files into directories
  - Assigning 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:

File Attributes: name, type, location, size, protection, creation time

- IBM mainframes implement files as a series of records or objects (structured)
- Unix: Implement files as a series of bytes (unstructured)
OS File Structures

2. Per-process file table - for each file:
   - pointers to location(s) of file in memory
   - pointers to location(s) of file on disk
   - file attributes, including ownership, protection information, access times, ...
   - open count

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

```
Rename()
Cgetattribute()  Getattribute()
Setlink()  Softlink()
Getattribute()  Hardlink()
```

Naming operations:
- attributes (owner, protection....)

Data operations:
- `Open`
- `Read`
- `Close`
- `Delete`

Common file operations:
- User Interface to the File System
File Operations: Creating a File

- Remove the file descriptor from the directory.
- Free the disk blocks used by the file.
- Find the directory containing the file.

File Operations: Deleting a File

- Unlink the link pointer (if the file). Macintosh/Windows opt 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 know what application to start, enables storage layer oblivious optimizations.
  - Advantages: better error detection; specialized default operations (double-clicking)

  - Optional file attributes: the type (word file, executable, etc.)
  - Add the file descriptor to the directory that contains the file.

  - Create a file descriptor for the file including name, location on disk, and all the
    attributes.

  - Allocate disk space (check disk quotas, permissions, etc.).
Read(field, size, bufferaddress) - sequential access

Read(field, from, size, bufferaddress) - random access

OS File Operations: Reading a File

Close(field)

File 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 file table.

- 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.
- Copy the file descriptor into the system-wide open file table.
- * Find the file.
- * 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.
Read/write.

Sequential: keep a pointer to the next byte in the file. Update the pointer on each

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.

Common file access patterns from the programmer’s perspective:

File Access Methods

Memory mapping a file

Seek just updates fp.

Write is similar to reads, but copies from the buffer to the file.

OS File Operations
There is one special root directory. Example: /usr/local/bin

Each directory contains "name, index" pairs in no particular order. The file

can write directories just like any other file, but only special system calls

zero directories on disk, just like files except the file descriptor for directories has a

modern operating systems)

• Multiple Level Directories - free structured name space (unix) and all other

Naming Strategies (continued)

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

• Statically because their disks were very small.

• Some early computers used this strategy. Early personal computers also used this

• 4. If one user uses a name, no one else can.

• 3. Directory contains "name, index" pairs.

• 2. Use a special area of disk to hold the directory.

• 1. Use a special area of disk to hold the directory.

- Single Level Directory: one name space for the entire disk, every name is unique.
- 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 tree name b in the directory, but its contents no longer exists.
- removing b does not affect a

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

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

Referential Naming

- Solution: no hard links to directories.
- delete the disk space.

Referential Naming

- Problem: use can create a 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

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

- Example: create a hard link from b to A.
- A hard link adds a second connection to a file.

Referential Naming
- 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 a list for each file with user name and type of access

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

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

**Protection**

**Directory Operations**

- Traverse the file system
- Rename a file
- List a directory: list all files (is 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
Summary of File System Functionality

- Fast access
- Persistence
- Protection
- Naming