Files and Directories

I/O Devices
System Architecture

• Buses: paths that exchange data among CPU, memory, and I/O devices
• I/O bus: a path that exchanges data between CPU and I/O devices
• Hierarchical structure
  • The faster a bus is, the shorter it is.
  • E.g., graphics is a high-performance device (nearer to CPU).

![Diagram of system architecture](image)
Canonical Device

- Hardware interface: allows software to control device’s operation
- Internal structure: hardware chips e.g., CPU, memory, *firmware* (an executable code that implements device interface)
The Canonical Protocol

• Three registers
  ▪ status: the current status of device
  ▪ command: ask device to perform a certain task
  ▪ data: to pass/get data to/from the device.

```c
while (STATUS == BUSY) {
    ; //wait until device is not busy
    write data to data register
    write command to command register
    Doing so starts the device and executes the command
    while (STATUS == BUSY) {
        ; //wait until device is done with your request
}
```

Polling wastes CPU time for slow devices.
Lowering CPU Overhead With Overhead

- OS makes an I/O request and puts the calling process to sleep.
- When the device finishes I/O, it interrupts CPU to run an interrupt handler.
- The handler reads data from the device and wakes the waiting process.
Lowering CPU Overhead With Overhead

- While a process is waiting for I/O, OS schedules another process on CPU.
- Interrupts improve CPU utilization (overlap computation and I/O).

Interrupts are not the best solution.
- Device finishes requests quickly
- Interrupts cause overheads: handling interrupts and switching between processes.

Optimize interrupts
- Device waits for other requests to complete.
- Coalesce multiple interrupts into a single interrupt.
Efficient Data Movement with DMA

• CPU runs instructions for transferring data to device
  ▪ Programmed I/O (PIO)
  ▪ Waste CPU time when copying data from memory to device

• Use Direct Memory Access (DMA)
  ▪ A programmed device
  ▪ Memory address, data length, and device name
Efficient Data Movement with DMA

- A process makes an I/O request.
- DMA controller copies data from memory to device.
- CPU is free to do other tasks.
Methods of Device Interaction

• I/O instructions
  ▪ Privileged machine instructions (e.g., in and out in x86)
  ▪ OS sends data to specific registers within some devices.
  ▪ Device registers are called I/O ports.
  ▪ E.g., out imm8, AL (imm8 is an I/O address)

• Memory-mapped I/O
  ▪ Device registers are mapped to a process address space.
  ▪ CPU executes standard memory instructions to read and write registers at their mapped addresses.

• Memory-mapped I/O is faster than I/O instructions.
Device Drivers

- OS interacts with different types of devices.
- OS needs to be device-neutral.
- Abstract the details of devices from OS.
- Device driver implements details.
  - Device driver is a piece of code within OS.
Files and Directories

File System Implementation
File System Aspects

- File system is an on-disk data structure.
  - Stores files
  - Disk is divided into partitions. A partition (volume) stores a file system.
  - A driver implements the data structure.
  - Structures are mapped on to the disk (mounting).

- Very Simple File System (VSFS)
- A simplified version of UNIX file system

- Two aspects of file system
  - Data structures: on-disk structures to store data
  - Access methods: routing system calls, e.g., open(), read(), and write(), to particular structures
Data Structures

• Divide disk into blocks
• Use one block size (4KB)
• Blocks are addressed from 0 to \(N-1\) (\(N\) is the number of blocks)

• Store user data in data region (e.g., files and directories)
Data Structures

- File system tracks information (metadata) about files.
  - E.g., a subset of data blocks that form a file, file size, access rights
- Metadata is stored in a structure called **inode**.
  - Inode table is an array of inodes.
  - 256-byte inode
  - 16 inodes per block; 80 inodes for 5 blocks
- Use **bitmap** for tracking free inodes and data blocks
  - A bit indicates whether an inode or a block is free.
- Superblock stores information about a certain file system.
  - E.g., number of inodes and data blocks, the start of inode table

![Diagram showing inodes and data region](image-url)
The Inode

- Index node (inode): array nodes are indexed.
- Each inode is identified by an i-number.
  - Used for computing the location (byte address) of an inode
- Find the byte address for the inode with i-number 32
  - Compute the offset into the inode table: 32 * sizeof(inode) = 8192 (8KB)
  - Add the offset to start address of inode table: 12KB + 8KB = 20KB
- Inodes are fetched using sectors (a block consists of sectors)
  - 512-byte sectors
  - Sector number: (20 * 1024)/512 = 40
The Inode

- Inode stores information about data blocks that belong to a file.
  - Each data block is identified by a direct pointer
- For a very big file, a large number pointers is required.
  - file size > data block size * number of direct pointers

- Inode has an indirect pointer
  - Points to another block that contains pointers

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>4KB block size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of blocks</td>
<td>4-byte disk address</td>
</tr>
<tr>
<td></td>
<td>Owner</td>
<td>1024 pointers</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>File size = (12 + 1024) * 4KB</td>
</tr>
<tr>
<td>Direct</td>
<td>pointers</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indirect pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Indirect block

...
A directory is a special type of file that
- is identified by an inode number.
- Has a list of contained files and their inode numbers.

<table>
<thead>
<tr>
<th>Name</th>
<th>Inode number</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>5</td>
</tr>
<tr>
<td>..</td>
<td>2</td>
</tr>
<tr>
<td>foo</td>
<td>12</td>
</tr>
</tbody>
</table>
Free Space management

• Track freed inodes or data blocks
  ▪ Use bitmaps
• After a new file is created, file system searches the **inode bitmap** (**i-bmap**) for a free inode.
  ▪ The allocated inode is marked as used (1) in **i-bmap**.
• If the file needs data blocks, file system searches the **data block bitmap** (**d-bmap**) for a free data block.
  ▪ Linux file system allocates a contiguous sequence of blocks.
  ▪ Increase performance
Reading a File from Disk

- Open a file, read it, and then close it.
  - E.g., open ("/foo/bar", O_RDONLY)
    - file size: 12KB (3 data blocks)
- write: updates the last access time of a data block.

<table>
<thead>
<tr>
<th>open(bar)</th>
<th>data bitmap</th>
<th>inode bitmap</th>
<th>root inode</th>
<th>foo inode</th>
<th>bar inode</th>
<th>root data</th>
<th>foo data</th>
<th>bar data[0]</th>
<th>bar data[1]</th>
<th>bar data[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fixed i-number

Pointer to data block
Writing to Disk

- Writing may allocate new block (five I/Os)
  - Read and write the data bitmap
  - Read and write an inode (writing the location of the new block)
  - Write to the data block
Caching and Buffering

• File reads and writes incur many I/Os from the slow disk
  – High performance cost

• A file open requires at least two reads for every directory in a pathname.
  – Read the inode and data block of each directory
  – Many reads for long pathnames (1/2/...100/file.txt)
Caching and Buffering

• Static partitioning
  – Fixed pages of memory are allocated for holding popular blocks.
  – Wastes space because file system may not use all of allocated pages

• Dynamic partitioning
  – Unified page cache: memory pages are allocated, on-demand, to processes or file system.
Caching and Buffering

• File open with caching
  – The first file open generate cache misses.
  – Subsequent file opens hit in the cache.

• Write buffering
  – Buffer a number of writes in memory to increase performance
  – Trade-off between performance and data loss