Storage Systems
(part 2)
File System Interface

• UNIX file open is a Swiss Army knife:
  – Open the file, return file descriptor
  – Options:
    • if file doesn’t exist, return an error
    • If file doesn’t exist, create file and open it
    • If file does exist, return an error
    • If file does exist, open file
    • If file exists but isn’t empty, nix it then open
    • If file exists but isn’t empty, return an error
    • ...
Interface Design Question

• Why not separate syscalls for open/create/exists?
  – Would be more modular!

if (!exists(name))
    create(name);    // can create fail?
fd = open(name);    // does the file exist?
Main Points

• Storage hardware
  – Disk scheduling
  – Flash memory
• File system usage patterns
• File system design
Track
Sector
Head
Arm
Arm Assembly
Platter
Surface
Surface
Motor
Spindle
Head
Arm
Sector
Track
Arm Assembly
Motor
Motor
Disk Performance

Disk Latency =

Seek Time + Rotation Time + Transfer Time
# Toshiba Disk (2008)

<table>
<thead>
<tr>
<th>Size</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Platters/Heads</td>
<td>2/4</td>
</tr>
<tr>
<td>Capacity</td>
<td>320 GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Spindle speed</td>
<td>7200 RPM</td>
</tr>
<tr>
<td>Average seek time read/write</td>
<td>10.5 ms/ 12.0 ms</td>
</tr>
<tr>
<td>Maximum seek time</td>
<td>19 ms</td>
</tr>
<tr>
<td>Track-to-track seek time</td>
<td>1 ms</td>
</tr>
<tr>
<td>Transfer rate (surface to buffer)</td>
<td>54–128 MB/s</td>
</tr>
<tr>
<td>Transfer rate (buffer to host)</td>
<td>375 MB/s</td>
</tr>
<tr>
<td>Buffer memory</td>
<td>16 MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical</td>
<td>16.35 W</td>
</tr>
<tr>
<td>Idle</td>
<td>11.68 W</td>
</tr>
</tbody>
</table>
Q&A

• How long to complete 500 random disk reads, in FIFO order?
  – 14 ms/read (avg seek + ½ rotation)
  – 70 random 512 byte reads/second

• How long to complete 500 sequential disk reads?
  – 16 ms/500 reads (avg seek + ½ rotation + transfer)
  – 60 random 250KB reads/second

• How large a transfer is needed to achieve 80% of the max disk transfer rate?
  – 10 MB
Disk Scheduling

- **SCAN**: move disk arm in one direction, until all requests satisfied, then reverse direction
Disk Scheduling

- CSCAN: move disk arm in one direction, until all requests satisfied, then start again from farthest request
Disk Scheduling

• R-CSCAN: CSCAN but take into account that short track switch is < rotational delay
Question

• How long to complete 500 random disk reads, in any order?
Question

• How long to complete 500 random disk reads, in any order?
  – Disk seek: 1ms (most will be short)
  – Rotation: 4.15ms
  – Transfer: 5-10usec

• Total: 500 * (1 + 4.15 + 0.01) = 2.2 seconds
  – Would be a bit shorter with R-CSCAN
  – vs. 7.3 seconds if FIFO order
Question

• How long to read all of the bytes off of a disk?
Question

• How long to read all of the bytes off of a disk?
  – Disk capacity: 320GB
  – Disk bandwidth: 54-128MB/s

• Transfer time =
  Disk capacity / average disk bandwidth
  ~ 3500 seconds (1 hour)
Flash Memory

Source

Control

Control Gate

Floating Gate

Drain

Source

Drain
Flash Memory

• Writes must be to “clean” cells; no update in place
  – Large block erasure required before write
  – Erasure block: 128 – 512 KB
  – Erasure time: Several milliseconds

• Write/read page (2-4KB)
  – 50-100 usec
Flash Drive (2011)

<table>
<thead>
<tr>
<th>Size</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>300 GB</td>
</tr>
<tr>
<td>Page Size</td>
<td>4KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth (Sequential Reads)</td>
<td>270 MB/s</td>
</tr>
<tr>
<td>Bandwidth (Sequential Writes)</td>
<td>210 MB/s</td>
</tr>
<tr>
<td>Read/Write Latency</td>
<td>75 μs</td>
</tr>
<tr>
<td>Random Reads Per Second</td>
<td>38,500</td>
</tr>
<tr>
<td>Random Writes Per Second</td>
<td>2,000 (2,400 with 20% space reserve)</td>
</tr>
<tr>
<td>Interface</td>
<td>SATA 3 Gb/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Endurance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Endurance</td>
<td>1.1 PB (1.5 PB with 20% space reserve)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Consumption Active/Idle</td>
<td>3.7 W / 0.7 W</td>
</tr>
</tbody>
</table>
Question

• Why are random writes so slow?
  – Random write: 2000/sec
  – Random read: 38500/sec
Flash Translation Layer

• Flash device firmware maps logical page # to a physical location
  – Move live pages as needed for erasure
    • Garbage collect empty erasure block by copying live pages to new location
  – Wear-levelling
    • Can only write each physical page a limited number of times
  – Avoid pages that no longer work

• Transparent to the device user
File System – Flash

• How does Flash device know which blocks are live?
  – Live blocks must be remapped to a new location during erasure

• TRIM command
  – File system tells device when pages are no longer in use
File System Workload

• File sizes
  – Are most files small or large?
  – Which accounts for more total storage: small or large files?
File System Workload

• File sizes
  – Are most files small or large?
    • SMALL
  – Which accounts for more total storage: small or large files?
    • LARGE
File System Workload

• File access
  – Are most accesses to small or large files?
  – Which accounts for more total I/O bytes: small or large files?
File System Workload

• File access
  – Are most accesses to small or large files?
    • SMALL
  – Which accounts for more total I/O bytes: small or large files?
    • LARGE
File System Workload

• How are files used?
  – Most files are read/written sequentially
  – Some files are read/written randomly
    • Ex: database files, swap files
  – Some files have a pre-defined size at creation
  – Some files start small and grow over time
    • Ex: program stdout, system logs
File System Design

• For small files:
  – Small blocks for storage efficiency
  – Concurrent ops more efficient than sequential
  – Files used together should be stored together

• For large files:
  – Storage efficient (large blocks)
  – Contiguous allocation for sequential access
  – Efficient lookup for random access

• May not know at file creation
  – Whether file will become small or large
  – Whether file is persistent or temporary
  – Whether file will be used sequentially or randomly
File System Design

• Data structures
  – Directories: file name -> file metadata
    • Store directories as files
  – File metadata: how to find file data blocks
  – Free map: list of free disk blocks

• How do we organize these data structures?
  – Device has non-uniform performance
Design Challenges

• Index structure
  – How do we locate the blocks of a file?

• Index granularity
  – What block size do we use?

• Free space
  – How do we find unused blocks on disk?

• Locality
  – How do we preserve spatial locality?

• Reliability
  – What if machine crashes in middle of a file system op?
## File System Design Options

<table>
<thead>
<tr>
<th></th>
<th>FAT</th>
<th>FFS</th>
<th>NTFS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index structure</strong></td>
<td>Linked list</td>
<td>Tree (fixed, assym)</td>
<td>Tree (dynamic)</td>
</tr>
<tr>
<td><strong>granularity</strong></td>
<td>block</td>
<td>block</td>
<td>extent</td>
</tr>
<tr>
<td><strong>free space allocation</strong></td>
<td>FAT array</td>
<td>Bitmap (fixed location)</td>
<td>Bitmap (file)</td>
</tr>
<tr>
<td><strong>Locality</strong></td>
<td>defragmentation</td>
<td>Block groups + reserve space</td>
<td>Extents Best fit defrag</td>
</tr>
</tbody>
</table>
Microsoft File Allocation Table (FAT)

• Linked list index structure
  – Simple, easy to implement
  – Still widely used (e.g., thumb drives)

• File table:
  – Linear map of all blocks on disk
  – Each file a linked list of blocks
FAT

• Pros:
  – Easy to find free block
  – Easy to append to a file
  – Easy to delete a file

• Cons:
  – Random access is very slow
  – Fragmentation
    • File blocks for a given file may be scattered
    • Files in the same directory may be scattered
    • Problem becomes worse as disk fills
Berkeley FFS (Fast File System)

- File metadata: inode table
  - similar to FAT table, except only for metadata
- File data: Assymetric tree
  - Small files: shallow tree
  - Large files: deep tree
  - Efficient storage for small files
  - Efficient lookup for random access in large files
FFS inode

• Metadata
  – File owner, access permissions, access times, ...

• Set of 12 data pointers
  – With 4KB blocks => max size of 48KB files

• Indirect block pointer
  – pointer to disk block of data pointers
  – 4KB block size => 1K data blocks => 4MB file

• Doubly indirect block pointer
  – 4GB file

• ...

FFS Locality

- File metadata spread throughout disk
  - Locate file metadata near file blocks
- First fit allocation
  - Small files fragmented, large files contiguous
- Block group allocation
  - Files in same directory located in nearby tracks