File Systems and Multiple Disks

- You can usually attach more than one disk to a machine
  - And often do
- Would it make sense to have a single file system span the several disks?
  - Considering the kinds of disk specific information a file system keeps
  - Like cylinder information
- Usually more trouble than it's worth
  - With the exception of RAID . . .
- Instead, put separate file system on each disk

How About the Other Way Around?

- Multiple file systems on one disk
- Divide physical disk into multiple logical disks
  - Often implemented within disk device drivers
  - Rest of system sees them as separate disk drives
- Typical motivations
  - Permit multiple OS to coexist on a single disk
    - E.g., a notebook that can boot either Windows or Linux
  - Separation for installation, back-up and recovery
    - E.g., separate personal files from the installed OS file system
  - Separation for free-space
    - Running out of space on one file system doesn't affect others

Working With Multiple File Systems

- So you might have multiple independent file systems on one machine
  - Each handling its own disk layout, free space, and other organizational issues
- How will the overall system work with those several file systems?
- Treat them as totally independent namespaces?
- Or somehow stitch the separate namespaces together?
- Key questions:
  1. How does an application specify which file it wants?
  2. How does the OS find that file?

Finding Files With Multiple File Systems

- Finding files is easy if there is only one file system
  - Any file we want must be on that one file system
  - Directories enable us to name files within a file system
- What if there are multiple file systems available?
  - Somehow, we have to say which one our file is on
- How do we specify which file system to use?
  - One way or another, it must be part of the file name
  - It may be implicit (e.g., same as current directory)
  - Or explicit (e.g., every name specifies it)
  - Regardless, we need some way of specifying which file system to look into for a given file name

Options for Naming With Multiple Partitions

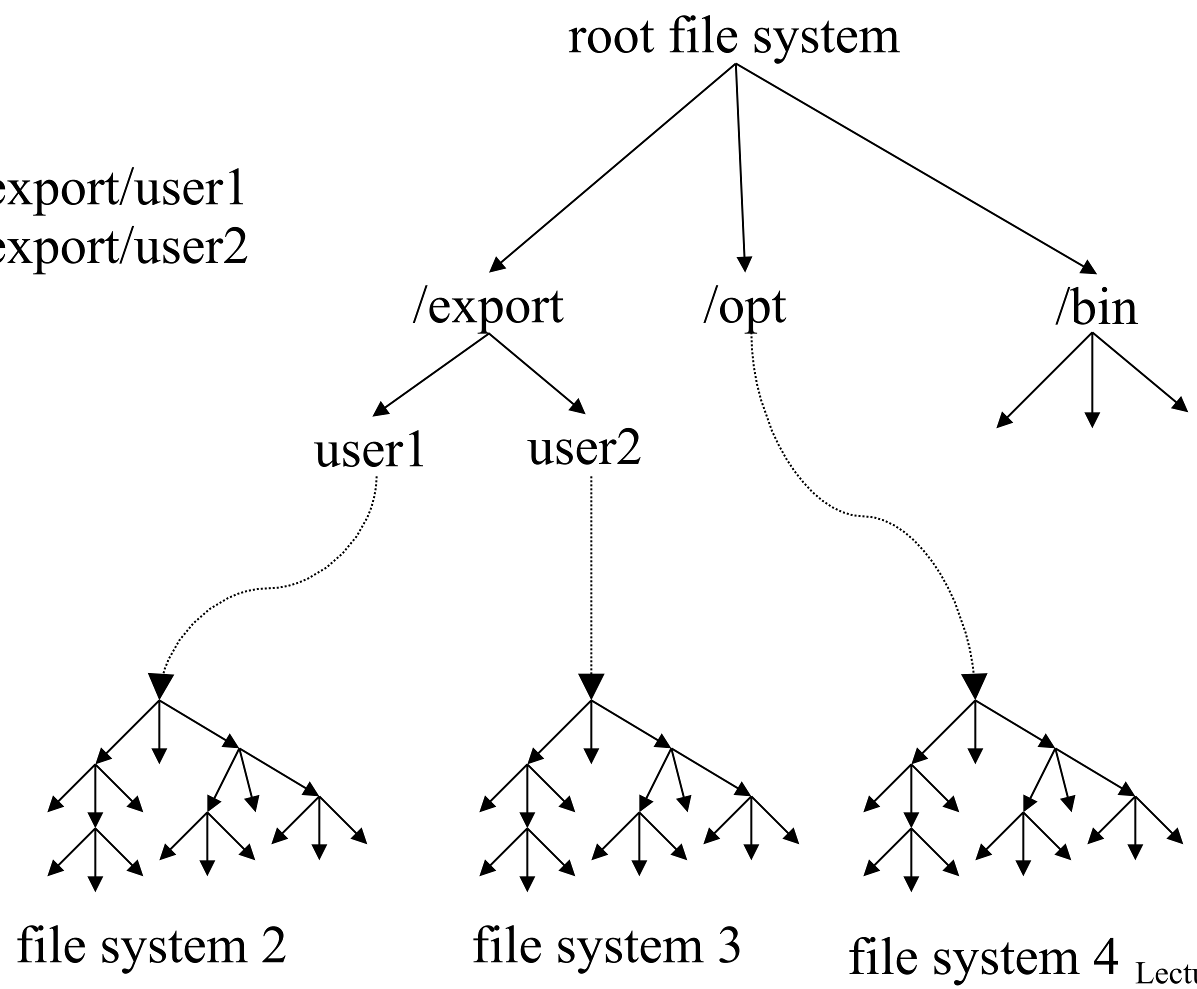
- Could specify the physical device it resides on
  - E.g., `/devices/pci/pci1000,4/disk/lun1/partition2`
    - That would get old real quick
- Could assign logical names to our partitions
  - E.g., “A:”, “C:”, “D:”
    - You only have to think physical when you set them up
    - But you still have to be aware multiple volumes exist
- Could weave a multi-file-system name space
  - E.g., Unix mounts

Unix File System Mounts

- Goal:
  - To make many file systems appear to be one giant one
  - Users need not be aware of file system boundaries
- Mechanism:
  - *Mount* device on directory
  - Creates a warp from the named directory to the top of the file system on the specified device
  - Any file name beneath that directory is interpreted relative to the root of the mounted file system

Unix Mounted File System Example

mount filesystem2 on /export/user1
mount filesystem3 on /export/user2
mount filesystem4 on /opt



How Does This Actually Work?

- Mark the directory that was mounted on
- When file system opens that directory, don't treat it as an ordinary directory
  - Instead, consult a table of mounts to figure out where the root of the new file system is
- Go to that device and open its root directory
- And proceed from there

File System Performance Issues

- Key factors in file system performance
  - Disk issues
    - Head movement
    - Block size
- Possible optimizations for file systems
  - Read-ahead
  - Delayed writes
  - Caching (general and special purpose)

File Systems and Disk Drives

- The physics of disk drives impact the performance of file systems
  - Which is unfortunate
- OS designers want to hide that impact
- To do so, they must hide variable disk delays
  - Preferably without making everything go at the slowest possible delay
- This requires many optimizations

Optimizing Disk I/O

- Don't start I/O until disk is on-cylinder or near sector
  - I/O ties up the controller, locking out other operations
  - Other drives seek while one drive is doing I/O
- Minimize head motion
  - Do all possible reads in current cylinder before moving
  - Make minimum number of trips in small increments
- Encourage efficient data requests
  - Have lots of requests to choose from
  - Encourage cylinder locality
  - Encourage largest possible block sizes
  - All by OS design choices, not influencing programs/users

Head Motion and File System Performance

- File system organization affects head motion
  - If blocks in a single file are spread across the disk
  - If files are spread randomly across the disk
  - If files and "meta-data" are widely separated
- All files are not used equally often
  - 5% of the files account for 90% of disk accesses
  - File locality should translate into head cylinder locality
- How do we use these factors to reduce head motion?

Ways To Reduce Head Motion

- Keep blocks of a file together
  - Easiest to do on original write
  - Try to allocate each new block close to the last one
  - Especially keep them in the same cylinder
- Keep metadata close to files
  - Again, easiest to do at creation time
- Keep files in the same directory close together
  - On the assumption directory implies locality of reference
- If performing compaction, move popular files close together

File System Performance and Block Size

- Larger block sizes result in efficient transfers
  - DMA is very fast, once it gets started
  - Per request set-up and head-motion is substantial
- They also result in internal fragmentation
  - Expected waste: ½ block per file
- As disks get larger, speed outweighs wasted space
  - File systems support ever-larger block sizes
- Clever schemes can reduce fragmentation
  - E.g., use smaller block size for the last block of a file

Read Early, Write Late

- If we read blocks before we actually need them, we don't have to wait for them
  - But how can we know which blocks to read early?
- If we write blocks long after we told the application it was done, we don't have to wait
  - But are there bad consequences of delaying those writes?
- Some optimizations depend on good answers to these questions

Read-Ahead

- Request blocks from the disk before any process asked for them
- Reduces process wait time
- When does it make sense?
  – When client specifically requests sequential access
  – When client seems to be reading sequentially
- What are the risks?
  – May waste disk access time reading unwanted blocks
  – May waste buffer space on unneeded blocks

Delayed Writes

- Don’t wait for disk write to complete to tell application it can proceed
- Written block is in a buffer in memory
- Wait until it’s “convenient” to write it to disk
  - Handle reads from in-memory buffer
- Benefits:
  - Applications don’t wait for disk writes
  - Writes to disk can be optimally ordered
  - If file is deleted soon, may never need to perform disk I/O
- Potential problems:
  - Lost writes when system crashes
  - Buffers holding delayed writes can’t be re-used

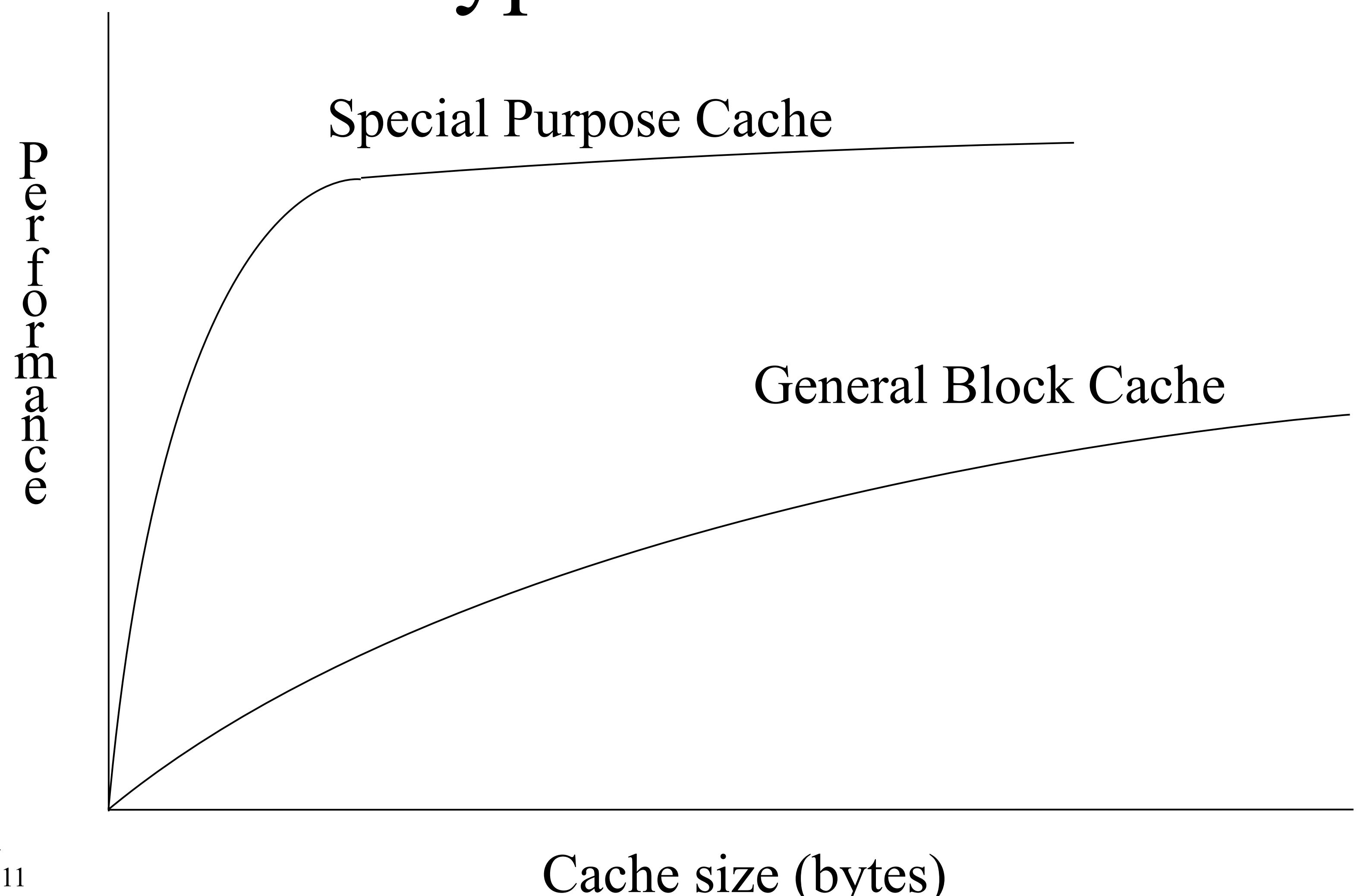
Caching and Performance

- Big performance wins are possible if caches work well
  - They typically contain the block you're looking for
- Should we have one big LRU cache for all purposes?
- Should we have some special-purpose caches?
  - If so, is LRU right for them?

Common Types of Disk Caching

- General block caching
  - Popular files that are read frequently
  - Files that are written and then promptly re-read
  - Provides buffers for read-ahead and deferred write
- Special purpose caches
  - Directory caches speed up searches of same dirs
  - Inode caches speed up re-uses of same file
- Special purpose caches are more complex
  - But they often work much better

Performance Gain For Different Types of Caches



Why Are Special Purpose Caches More Effective?

- They match caching granularity to their need
  - E.g., cache inodes or directory entries
  - Rather than full blocks
- Why does that help?
- Consider an example:
  - A block might contain 100 directory entries, only four of which are regularly used
  - Caching the other 96 as part of the block is a waste of cache space
  - Caching 4 entries allows more popular entries to be cached
  - Tending to lead to higher hit ratios

File Systems Reliability

- File systems are meant to store data persistently
- Meaning they are particularly sensitive to errors that screw things up
  - Other elements can sometimes just reset and restart
  - But if a file is corrupted, that's really bad
- How can we ensure our file system's integrity is not compromised?

Causes of System Data Loss

- OS or computer stops with writes still pending
  - .1-100/year per system
- Defects in media render data unreadable
  - .1 – 10/year per system
- Operator/system management error
  - .01-.1/year per system
- Bugs in file system and system utilities
  - .01-.05/year per system
- Catastrophic device failure
  - .001-.01/year per system

Dealing With Media Failures

- Most media failures are for a small section of the device, not huge extents of it
- Don't use known bad sectors
  - Identify all known bad sectors (factory list, testing)
  - Assign them to a “never use” list in file system
  - Since they aren't free, they won't be used by files
- Deal promptly with newly discovered bad blocks
  - Most failures start with repeated “recoverable” errors
  - Copy the data to another block ASAP
  - Assign new block to file in place of failing block
  - Assign failing block to the “never use” list

Problems Involving System Failure

- Delayed writes lead to many problems when the system crashes
- Other kinds of corruption can also damage file systems
- We can combat some of these problems using ordered writes
- But we may also need mechanisms to check file system integrity
  – And fix obvious problems

Deferred Writes – Promise and Dangers

- Deferring disk writes can be a big performance win
  - When user updates files in small increments
  - When user repeatedly updates the same data
- It may also make sense for meta-data
  - Writing to a file may update an indirect block many times
  - Unpacking a zip creates many files in same directory
  - It also allocates many consecutive inodes
- But deferring writes can also create big problems
  - If the system crashes before the writes are done
  - Some user data may be lost
  - Or even some meta-data updates may be lost

Performance and Integrity

- It is very important that file system be fast
  - File system performance drives system performance
- It is absolutely vital that they be robust
  - Files are used to store important data
    - E.g., student projects, grades, video games, …
- We must know that our files are safe
  - That the files will not disappear after they are written
  - That the data will not be corrupted

Deferred Writes – A Worst Case Scenario

- Process allocates a new block for file A
  - We get a new block (x) from the free list
  - We write the updated inode for file A
    - Including a pointer to x
  - We defer free-list write-back (which happens all the time)
- The system crashes, and after it reboots
  - A new process wants a new block for file B
  - We get block x from the (stale) free list
- Two different files now contain the same block
  - When file A is written, file B gets corrupted
  - When file B is written, file A gets corrupted

Ordering Writes

- Many file system corruption problems can be solved by carefully ordering related writes
- Write out data before writing pointers to it
  - Unreferenced objects can be garbage collected
  - Pointers to incorrect data/meta-data are much more serious
- Write out deallocations before allocations
  - Disassociate resources from old files ASAP
  - Free list can be corrected by garbage collection
  - Improperly shared blocks more serious than unlinked ones
- But it may reduce disk I/O efficiency
  - Creating more head motion than elevator scheduling

Backup – The Ultimate Solution

- All files should be regularly backed up
- Permits recovery from catastrophic failures
- Complete vs. incremental back-ups
- Desirable features
  - Ability to back-up a running file system
  - Ability to restore individual files
  - Ability to back-up w/o human assistance
- Should be considered as part of FS design
  - I.e., make file system backup-friendly