SCSI overview

- **SCSI domain** consists of devices and an SDS
  - Devices: host adapters & SCSI controllers
  - *Service Delivery Subsystem* connects devices–e.g., SCSI bus

- **SCSI-2 bus (SDS) connects up to 8 devices**
  - Controllers can have > 1 “logical units” (LUNs)
  - Typically, controller built into disk and 1 LUN/target, but “bridge controllers” can manage multiple physical devices

- **Each device can assume role of initiator or target**
  - Traditionally, host adapter was initiator, controller target
  - Now controllers act as initiators (e.g., COPY command)
  - Typical domain has 1 initiator, ≥ 1 targets
SCSI requests

- A request is a command from initiator to target
  - Once transmitted, target has control of bus
  - Target may disconnect from bus and later reconnect
    (very important for multiple targets or even multitasking)

- Commands contain the following:
  - Task identifier—initiator ID, target ID, LUN, tag
  - Command descriptor block—e.g., read 10 blocks at pos. \( N \)
  - Optional task attribute—SIMPLE, ORDERD, HEAD OF QUEUE
  - Optional: output/input buffer, sense data
  - Status byte—GOOD, CHECK CONDITION, INTERMEDIATE, …
Executing SCSI commands

- Each LUN maintains a queue of *tasks*
  - Each task is DORMANT, BLOCKED, ENABLED, or ENDED
  - SIMPLE tasks are dormant until no ordered/head of queue
  - ORDERED tasks dormant until no HoQ/more recent ordered
  - HoQ tasks begin in enabled state

- Task management commands available to initiator
  - Abort/terminate task, Reset target, etc.

- Linked commands
  - Initiator can link commands, so no intervening tasks
  - E.g., could use to implement atomic read-modify-write
  - Intermediate commands return status byte INTERMEDIATE
SCSI exceptions and errors

• After error stop executing most SCSI commands
  - Target returns with CHECK CONDITION status
  - Initiator will eventually notice error
  - Must read specifics w. REQUEST SENSE

• Prevents unwanted commands from executing
  - E.g., initiator may not want to execute 2nd write if 1st fails

• Simplifies device implementation
  - Don’t need to remember more than one error condition

• Same mechanism used to notify of media changes
  - I.e., ejected tape, changed CD-ROM
But back in the 80s…

- Disks spin at 3,600 RPM
  - 17 ms/Rotation (vs. 4 ms on fastest disks today)

- Fixed # sectors/track (no zoning)

- Head switching free (?)

- Requests issued one at a time
  - No caching in disks
  - Head must pass over sector after getting a read
  - By the time OS issues next request, too late for next sector

- Slower CPUs, memory
  - Noticeable cost for block allocation algorithms
Original Unix file system

- Each FS breaks partition into three regions:
  - Superblock (parameters of file system, free ptr)
  - Inodes – type/mode/size + ptr to data blocks
  - File and directory data blocks

- All data blocks 512 bytes

- Free blocks kept in a linked list
Inodes

inode

metadata
data ptr
data ptr
indirect ptr
double indir

indirect block
data ptr
data ptr

directory

name
i-number

contents

data
data
data

data
Problems with original FS

- FS never transfers more than 512 bytes/disk access
- After a while, allocation essentially random
  - Requires a random seek every 512 bytes of file data
- Inodes far from both directory data and file data
- Within a directory, inodes not near each other
- Usability problems:
  - File names limited to 14 characters
  - No way to update file atomically & guarantee existence after crash
Fast file system

- New block size must be at least 4K
  - To avoid wasting space, use “fragments” for ends of files
- Cylinder groups avoid spread inodes around disk
- Bitmaps replace free list
- FS reserves space to improve allocation
  - Tunable parameter, default 10%
  - Only superuser can use space when over 90% full
FFS superblock

- Contains file system parameters
  - Disk characteristics, block size, CG info
  - Information necessary to get inode given i-number
- Replicated once per cylinder group
  - At shifting offsets, so as to span multiple platters
  - Contains magic to find replicas if 1st superblock dies
- Contains non-replicated “summary info”
  - # blocks, fragments, inodes, directories in FS
  - Flag stating if FS was cleanly unmounted
Cylinder groups

- Groups related inodes and their data
- Contains a number of inodes (set when FS created)
  - Default one inode per 2K data

- Contains file and directory blocks

- Contains bookkeeping information
  - Block map – bit map of available fragments
  - Summary info within CG – # free inodes, blocks/frags, files, directories
  - # free blocks by rotational position (8 positions)
Inode allocation

• Allocate inodes in same CG as directory if possible

• New directories put in new cylinder groups
  - Consider CGs with greater than average # free inodes
  - Chose CG with smallest # directories

• Within CG, inodes allocated randomly (next free)
  - Would like related inodes as close as possible
  - OK, because one CG doesn’t have that many inodes
Fragment allocation

- Allocate space when user writes beyond end of file
- Want last block to be a fragment if not full-size
  - If already a fragment, may contain space for write – done
  - Else, must deallocate any existing fragment, allocate new
- If no appropriate free fragments, break full block
- Problem: Slow for many small writes
  - (Partial) solution: new stat struct field `st_blksize`
  - Tells applications file system block size
  - `stdio` library can buffer this much data
Block allocation

- **Try to optimize for sequential access**
  - If available, use rotationally close block in same cylinder
  - Otherwise, use block in same CG
  - If CG totally full, find other CG with quadratic hashing
  - Otherwise, search all CGs for some free space

- **Problem: Don’t want one file filling up whole CG**
  - Otherwise other inodes will have data far away

- **Solution: Break big files over many CGs**
  - But large extents in each CGs, so sequential access doesn’t require many seeks
Directories

• Inodes like files, but with different type bits

• Contents considered as 512-byte chunks

• Each chunk has direct structure(s) with:
  - 32-bit inumber
  - 16-bit size of directory entry
  - 8-bit file type (NEW)
  - 8-bit length of file name

• Coalesce when deleting
  - If first direct in chunk deleted, set inumber = 0

• Periodically compact directory chunks
Updating FFS for the 90s

- No longer want to assume rotational delay
  - With disk caches, want data contiguously allocated

- Solution: Cluster writes
  - FS delays writing a block back to get more blocks
  - Accumulates blocks into 64K clusters, written at once

- Allocation of clusters similar to fragments/blocks
  - Summary info
  - Cluster map has one bit for each 64K if all free

- Also read in 64K chunks when doing read ahead
Dealing with crashes

- Suppose all data written asynchronously

- Delete/truncate a file, append to other file, crash
  - New file may reuse block from old
  - Old inode may not be updated
  - Cross-allocation!
  - Often inode with older mtime wrong, but can’t be sure

- Append to file, allocate indirect block, crash
  - Inode points to indirect block
  - But indirect block may contain garbage
Ordering of updates

- Must be careful about order of updates
  - Write new inode to disk before directory entry
  - Remove directory name before deallocating inode
  - Write cleared inode to disk before updating CG free map

- Solution: Many metadata updates synchronous
  - Of course, this hurts performance
  - E.g., untar much slower than disk b/w

- Note: Cannot update buffers on the disk queue
Fixing corruption – fsck

- **Summary info usually bad after crash**
  - Scan to check free block map, block/inode counts

- **System may have corrupt inodes (not simple crash)**
  - Bad block numbers, cross-allocation, etc.
  - Do sanity check, clear inodes with garbage

- **Fields in inodes may be wrong**
  - Count number of directory entries to verify link count, if no entries but count ≠ 0, move to lost+found
  - Make sure size and used data counts match blocks

- **Directories may be bad**
  - Holes illegal, . and .. must be valid, …
  - All directories must be reachable